CONTROLLER DEPENDABILITY ANALYSIS BY PROBABILISTIC MODEL CHECKING

Marta Kwiatkowska, Gethin Norman and David Parker [1]

School of Computer Science, University of Birmingham, Birmingham, B15 2TT, United Kingdom

Abstract: This paper demonstrates how probabilistic model checking, a formal verification method for the analysis of systems which exhibit stochastic behaviour, can be applied to the study of dependability properties of software-based control systems. By using existing formalisms and tool support from this area, it is possible to construct large and complex Markov models from an intuitive high-level description and to take advantage of the efficient implementation techniques which have been developed for these tools. This paper provides an overview of probabilistic model checking and of the tool PRISM which supports these techniques. It illustrates the applicability of the approach through the use of a case study and demonstrates that a wide range of useful dependability properties can be analysed in this way.

Keywords: Formal verification, performance analysis, Markov models

1. INTRODUCTION

Industrial use of software-based control systems is now increasingly widespread. Programmable controllers can be found in domains as diverse as the manufacturing, transport, energy and power industries. In many of these cases, the safety and reliability of the system is a crucial issue. This fact is evidenced, for example, by the existence, and indeed prominence, of the IEC 61508 and ANSI/ISA S84 standards, which include strict guidelines as to the functional safety of such systems.

A key feature of these standards is the definition of Safety Integrity Levels (SILs). To adhere to a particular SIL, it is necessary to accurately quantify the probability or rate of all safety-related faults which can occur in the system under study. Two common techniques for performing such dependability analysis are reliability block diagrams and fault tree analysis. Both, however, are relatively simplistic. For example, it is often necessary to assume that the probability of a certain fault arising is independent of the occurrence of other failures in the system, whereas in practice, this may be unrealistic and can lead to inaccuracies in the analysis.

A more sophisticated approach which resolves this problem is Markov modelling, where a more accurate, state-based model of the system is derived and analysed. This does, however, have two disadvantages. Firstly, it is more complex for the end-user to implement. Secondly, the size of model required to represent a system accurately has a tendency to increase exponentially. This is a phenomenon often known as the state space explosion problem. Not only does it make the process more expensive in terms of the time, computing power and memory required, it can make the reliability analysis infeasible.

[1] Corresponding author. Email: dxp@cs.bham.ac.uk
Phone: +44 (0)121 41 44793 Fax: +44 (0)121 41 44281

This paper shows how formal verification methods can be applied to the problem of analysing the dependability of controller-based systems. In particular, the focus is on probabilistic model checking, an automatic technique for checking whether or not probabilistic models satisfy certain specifications. It is shown how existing formalisms and tools from this area can facilitate the specification and analysis of Markov models. In addition, the efficient implementation techniques which have been integrated into these tools can be used in an attempt to curb the effects of state space explosion.

The remainder of this paper is organised as follows. Section 2 gives an introduction to probabilistic model checking and Section 3 describes the tool PRISM, which implements these techniques. Section 4 introduces a case study of a simple control system and Section 5 presents results obtained from an analysis of this case study using PRISM. Section 6 concludes the paper.

2. PROBABILISTIC MODEL CHECKING

Model checking (Clarke et al., 1999) is a successful and well established technique for formally verifying the correctness of finite-state systems. In recent years, such methods have become increasingly prevalent in industry, with large companies such as IBM and Intel devoting considerable resources to this area. Model checking involves the construction of a formal model of the real-life system which is to be verified. This is usually a labelled state transition system, which represents all the possible configurations which the system can be in and all the transitions which can occur between them. The properties of the system to be verified are then also formally specified, usually as formulas of a temporal logic, and passed to a model checker, which automatically determines whether or not each property is satisfied via a systematic exploration of the model. In the case of a negative result, a counterexample is often generated: an explicit trace of the system's behaviour which illustrates why a property was not satisfied.

An extension of this approach which has seen a significant amount of development of late is *probabilistic model checking*, a technique which permits automatic formal verification of systems which exhibit stochastic behaviour. Potential candidates for such analysis include randomised algorithms, which use probabilistic choices or electronic coin flipping, and unreliable or unpredictable processes, such as fault-tolerant systems or communication networks.

Probabilistic model checking is again based on the construction and analysis of a formal model of the system. In this case, the model is enriched with probabilistic information, typically by labelling each transition of the model with information about the likelihood that it will occur. This paper focuses on a class of models called *continuous-time Markov chains* (CTMCs). A CTMC comprises a set of states S and a transition rate matrix $\mathbf{R} : S \times S \rightarrow \mathbb{R}_{\geq 0}$. The *rate* $\mathbf{R}(s, s')$ defines the delay before which a transition between states s and s' is enabled. The delay is sampled from a negative exponential distribution with parameter equal to this rate, i.e. the probability of the transition being enabled within t time units is $1 - e^{-\mathbf{R}(s,s') \cdot t}$. When $\mathbf{R}(s, s') > 0$ for more than one state s', there is a *race* between the outgoing transitions of s. More precisely, the probability of moving from state s to s' in a single step is the probability that this transition is enabled first (i.e. that the delay of this transition finishes before the delays of all other transitions leaving s). Exponentially distributed delays are often suitable for modelling component lifetimes and inter-arrival times. Furthermore, they can be used to approximate more complex probability distributions.

Other models commonly used for probabilistic model checking are discrete-time Markov chains (DTMCs), which specify the probability of moving between states in discrete time-steps, and Markov decision processes (MDPs), which can model systems which exhibit both probabilistic and nondeterministic behaviour. The latter are useful for representing systems which comprises a number of probabilistic processes operating asynchronously in parallel or systems for which some parameters are unknown.

Given a probabilistic model, the next step is to specify its required properties. Traditionally, in the model checking paradigm, properties are expressed using temporal logic, which provides a concise and unambiguous specification. In this paper, the temporal logic CSL (Continuous Stochastic Logic) (Aziz et al., 1996; Baier et al., 1999) which is designed for specifying properties of CTMCs is used. For brevity, the full syntax and semantics of the logic are not presented here. Below are a number of illustrative examples with their natural language translation:

- $\mathcal{P}_{\geq 0.98}[\ \Diamond\ \mathit{complete}\ ]$ – "the probability of the system eventually completing its execution successfully is at least 0.98"
- $\mathcal{P}_{<0.01}[\ \Diamond^{[1,1]}\ \mathit{queue\_size}{=}\mathit{max}\ ]$ – "the probability of the queue being full after exactly one hour is less than 0.01"
- $\mathit{shutdown} \Rightarrow \mathcal{P}_{\geq 0.95}[\ \neg \mathit{fail}\ \mathcal{U}^{\leq 200}\ \mathit{up}\ ]$ – "once a shutdown has occurred, with probability 0.95 or greater, the system will successfully

recover within 200 hours and without any further failures occurring"

- $\mathcal{S}_{>0.75}[\ num\_sensors \geq min\ ]$ – "in the long-run, the probability that an adequate number of sensors are operational is greater than 0.75"

Note that the use of probability bounds (≥ 0.98, < 0.01, ≥ 0.95, > 0.75) ensures that the properties above constitute questions which can be verified either to be true or false, as is traditionally the case in formal verification. In practice, though, it is often more useful to request the actual values, writing for example:

- $\mathcal{P}_{=?}[\ \Diamond^{\leq T} shutdown\ ]$ – "what is the probability that the system shuts down by time T?"

Furthermore, the most useful way to analyse the model and to gain insights into its reliability may be to compute and plot such a value as some parameter is varied (e.g. T in the formula above, or a constant in the model itself).

Additional properties can be specified by adding the notion of *rewards*. Each state (and/or transition) of the model is assigned a real-valued reward, allowing queries such as:

- $\mathcal{R}_{=?}[\ \Diamond success\ ]$ – "what is the expected reward accumulated before the system successfully terminates?"

Rewards can be used to specify a wide range of measures of interest, for example, the number of correctly delivered messages or the time that the system is operational. Of course, conversely, the rewards can be considered as costs, such as power consumption, expected number of failures, etc. Suitable temporal logics for expressing cost- and reward-based properties can be found in e.g. (Baier et al., 2000; de Alfaro, 1997).

A *probabilistic model checker* applies algorithmic techniques to first construct, from a high-level description, a probabilistic model of the system under study and then analyse the constructed model to determine whether its specifications are satisfied. Typically, this involves computation of one or more probabilities or performance measures. The operations required are graph-based analysis and numerical computation methods, for example, solving linear equation systems or linear optimisation problems. Algorithms for model checking the logic CSL can be found in (Baier et al., 2003). For more detailed information about probabilistic model checking in general, see for example (Rutten et al., 2004).

3. PRISM

PRISM (Kwiatkowska et al., 2004) is a probabilistic model checker developed at the University of Birmingham. It supports construction and analysis of the three types of probabilistic model mentioned in the previous section: continuous-time Markov chains (CTMCs), discrete-time Markov chains (DTMCs) and Markov decision processes (MDPs).

Probabilistic models to be analysed in PRISM are specified in the PRISM language, which is based on the Reactive Modules formalism of Alur and Henzinger (Alur and Henzinger, 1999). A model is described as a number of *modules*, each of which corresponds to a component of the real-life system. Each module has a set of finite-ranged *variables*. These determine the possible states of each module. The whole model is constructed as the parallel composition of these modules.

The behaviour of an individual module is specified by a set of guarded commands. For a CTMC, as is the case here, a command takes the form:

$$[]\ \texttt{<guard>}\ \rightarrow\ \texttt{<rate>}\ :\ \texttt{<action>}\ ;$$

The *guard* is a predicate over the variables of all the modules in the model. The *update* comprises `<rate>`, an expression which evaluates to a positive real number, and `<action>`, which describes a transition of the module in terms of how its variables should be updated. The interpretation of the command is that if the guard is satisfied, then the module can make the corresponding transition with that rate (see Section 2 for a definition of rate). A simple command for a module with one variable x might be:

$$[]\ (x = 0)\ \rightarrow\ 4.5\ :\ (x' = x + 1)\ ;$$

which states that if x is 0, it is incremented by one (primed variables such as x' denote the new value of a variable x). In this example, the action of x being incremented occurs with rate 4.5 (i.e. the delay before this transition is completed is sampled from a negative exponential distribution with parameter 4.5).

The overall functionality of the PRISM tool is as follows. First, it reads and parses a system description in the PRISM language. It then constructs, from this, the corresponding probabilistic model, in this case a CTMC (although the same description language can be used for both DTMCs and MDPs). PRISM also computes the set of all states which are reachable from the initial state and identifies any deadlock states (i.e. reachable states with no outgoing transitions). If required, the transition matrix of the probabilistic model constructed can be exported for use in another

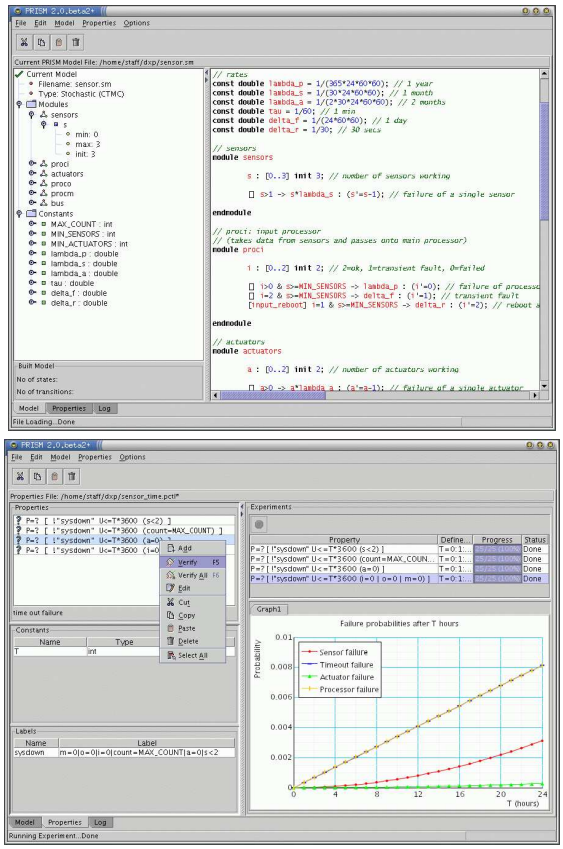


Fig. 1. Screenshots of the PRISM tool running

tool. Typically, though, PRISM then parses one or more temporal logic properties (e.g. in CSL) and performs model checking, determining whether the model satisfies each property. The graphical user interface of the tool also allows automatic construction of graphs to visualise results more easily. Figure 1 shows two screenshots of the tool running.

Another important feature of the tool is its implementation. PRISM is a "symbolic" model checker, meaning that it has been developed using data structures based on binary decision diagrams (BDDs) (Bryant, 1995). BDDs are reduced, directed acyclic graphs which can be very effective for compact storage of models which exhibit structure and regularity (e.g. those which have been specified in a high-level description language such as the PRISM language). In practice, this allows construction and analysis of larger models than would otherwise be possible. It also allows efficient BDD-based algorithms developed for non-probabilistic model checking to be easily integrated.

PRISM has already been used to analyse a wide range of case studies from many different areas. These include real-time probabilistic communication protocols, e.g. Bluetooth (Duflot et al., 2004), IEEE 1394 FireWire (Kwiatkowska et al., 2003b) and Zeroconf (Kwiatkowska et al., 2003a); and randomised security protocols, e.g.

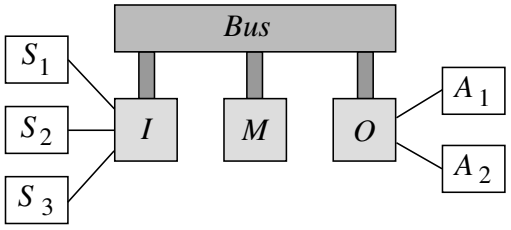


Fig. 2. Case study: overall system structure

Component	Mean time to failure
Sensor	1 month
Actuator	2 months
Processor	1 year (permanent fault)
Processor	1 day (transient fault)
Delay	**Mean time**
Timer cycle	1 minute
Processor reboot	30 seconds

Fig. 3. Case study: statistics for component reliability and timings of other system delays

for anonymity (Shmatikov, 2004), contract signing (Norman and Shmatikov, 2003) and non-repudiation (Lanotte et al., 2004). Other application domains which have been analysed with the tool include dynamic power management (Norman et al., 2002), fault-tolerant architectures (Norman et al., 2005), and randomised distributed algorithms, e.g. (Kwiatkowska et al., 2001; Kwiatkowska and Norman, 2002).

The tool and its source code can be freely downloaded under the GNU General Public License from www.cs.bham.ac.uk/~dxp/prism. Tool documentation, relevant research papers, and detailed information about a wide range of previous case studies can also be found here.

4. CASE STUDY

To illustrate the applicability of probabilistic model checking and the PRISM tool to the process of dependability analysis, this section presents a small case study. It models a relatively simple control system, closely based on the one presented in (Muppala et al., 1994), the structure of which is shown in Figure 2. The system comprises an input processor (I) which reads and processes data from three sensors (S_1, S_2 and S_3). It is then polled by a main processor (M) which, in turn, passes instructions to an output processor (O). This process occurs cyclically, with the length of the cycle controlled by a timer in the main processor. The output processor takes the instructions it receives and uses them to control two actuators (A_1 and A_2). Finally, a bus connects the three processors together. A concrete example of such a system might be a gas boiler, where the sensors are thermostats and the actuators are valves.

```
// a sensor fails on average once a month
const double λs = 1/(30 · 24 · 60 · 60);

module sensors

    // number of sensors working
    s : [0..3] init 3;

    // failure of a single sensor
    [] s > 0 → s · λs : (s′ = s − 1);

endmodule
```

Fig. 4. PRISM code for the sensors

Any of the three sensors can fail, but they are used in triple modular redundancy: the input processor can determine sufficient information to proceed provided two of the three are functional. If more than one becomes unavailable, the processor reports this fact to the main processor and the system is shut down. In similar fashion, it is sufficient for only one of the two actuators to be working, but if this is not the case, the output processor tells the main processor to shut the system down. The I/O processors themselves can also fail. 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. In either case, if the I or O processor is unavailable and this leads to M being unable either to read data from I or send instructions to O, then M is forced to skip the current cycle. If M detects that the number of consecutive cycles skipped exceeds a limit, K, then it assumes that either I or O has failed and shuts the system down. Unless specified otherwise, K is assumed to take the value 2. Lastly, the main processor can also fail, in which case the system is automatically shut down.

The mean time to failure for each component of the system is shown in Figure 3. Mean times for other delays in the system, namely the time for the main processor to complete a cycle and for the I/O processors to reboot are also given. It is assumed that all these delays are distributed exponentially; hence the system can be modelled as a CTMC.

The PRISM model of the system comprises 6 modules, one for the sensors, one for the actuators, one for each processor and one for the connecting bus. Figure 4 shows the section of the PRISM language description which models the sensors. This constitutes a single module *sensors* with an integer variable s representing the number of sensors currently working. The module's behaviour is described by one guarded command which represents the failure of a single sensor. Its guard "$s > 0$" states this can occur at any time, except when all sensors have already failed. The action $(s' = s - 1)$ simply decrements the counter of functioning sensors. The rate of this action is $s \cdot \lambda_s$, where λ_s is the rate for a single sensor and s is the PRISM variable referred to previously which denotes the number of active sensors.

```
const double λp = 1/(365 · 24 · 60 · 60);
const double δf = 1/(24 · 60 · 60);
const double δr = 1/30;

module proci

    // state: 2=ok, 1=transient fault, 0=failed
    i : [0..2] init 2;

    // failure of processor
    [] i > 0 ∧ s ≥ 2 → λp : (i′ = 0);
    // transient fault
    [] i = 2 ∧ s ≥ 2 → δf : (i′ = 1);
    // reboot after transient fault
    [input_reboot] i = 1 → δr : (i′ = 2);

endmodule
```

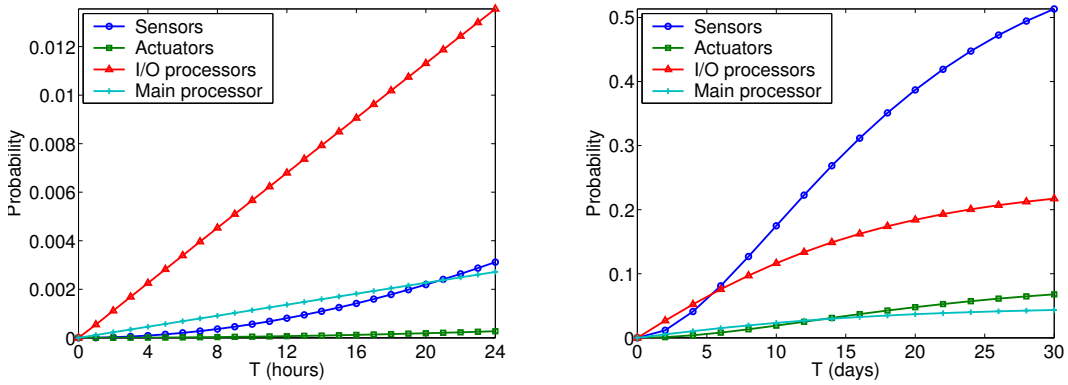
Fig. 5. PRISM code for the input processor

Figure 5 shows a second module which is the PRISM language description of the input processor. The module has a single variable i with range $\{0, 1, 2\}$ which indicates which of the three possible states the processor is in, i.e. whether it is working, is recovering from a transient fault, or has failed. The three guarded commands in the module correspond, respectively, to the processor failing, suffering a transient fault, and rebooting. The commands themselves are fairly self explanatory. Two points of note are as follows. Firstly, the guards of these commands can refer to variables from other modules, as evidenced by the use of $s \geq 2$. This is because the input processor ceases to function once it has detected that less than two sensors are operational. Secondly, the last command contains an additional label *input_reboot*, placed between the square brackets at the start of the command. This is used for synchronising actions between modules, i.e. allowing two or more modules to make transitions simultaneously. Here, this is used to notify the main processor of the reboot as soon as it occurs.

The full version of the PRISM code for this case study is available from the tool website.

5. RESULTS

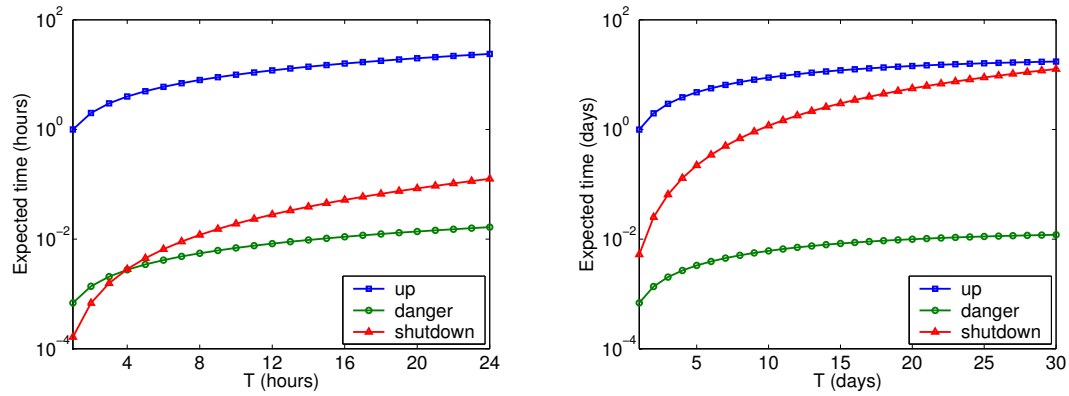
PRISM has been used to construct the CTMC representing the control system described in the previous section and to analyse a number of dependability properties using probabilistic model checking. First, the probability of the system shutting itself down is considered. Note that there are four distinct types of failure which can cause a shutdown: faults in (1) the sensors (2) the actuators (3) the input/output processors (4) the main



(a) First 24 hours of system operation

(b) First 30 days of system operation

Fig. 6. The probability that each of the four possible failure types is the cause of system shutdown



(a) First 24 hours of system operation

(b) First 30 days of system operation

Fig. 7. The expected amount of time spent in each of the three states: "up", "shutdown" and "danger"

processor. The following CSL property is used to analyse how likely each of these is to be the cause of the shutdown, as time passes:

- $\mathcal{P}_{=?}[\ \neg \mathit{shutdown}\ \mathcal{U}^{\leq T}\ \mathit{fail}_j\ ]$

where $j = 1 \ldots 4$, refers to one of the four failures above and *shutdown* denotes the fact that the system has shut down, i.e. a failure has occurred.

The atomic propositions, such as *shutdown* and fail_j, which make up the property are in practice predicates over the variables from the PRISM language description. For example, fail_1, the failure of more than one sensor, is specified as follows:

- $\mathit{fail}_1 := s < 2\ \wedge\ i = 2$

meaning that the number of working sensors has dropped below 2 and the input processor is functioning (and so can report the failure). Similarly, *shutdown*, indicating that the system has shut down, is specified by:

- $\mathit{shutdown} := \mathit{fail}_1 \vee \mathit{fail}_2 \vee \mathit{fail}_3 \vee \mathit{fail}_4$

meaning that one of the failures has occurred.

The property $\mathcal{P}_{=?}[\ \neg \mathit{shutdown}\ \mathcal{U}^{\leq T}\ \mathit{fail}_j\ ]$ denotes the probability that failure j occurs within T time units and no other failure has occurred before failure j occurs. Note that, for example, if an actuator fails, the sensors, unaware of this, will continue to operate and may subsequently fail. Hence, it is necessary to determine the likelihood of each failure occurring, before any of the others do. This is a good illustration of how non-trivial properties can be captured using temporal logic.

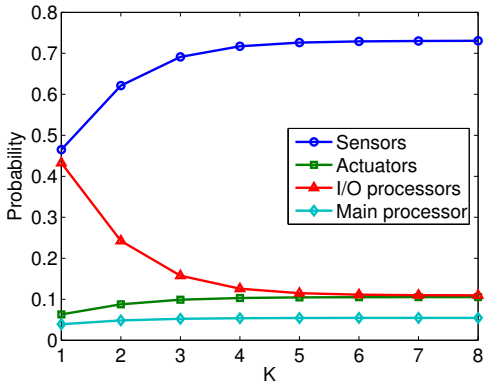
Figure 6 plots the results of an analysis of this property with PRISM over two ranges of values for the time parameter T: the first 24 hours and the first 30 days of operation. It can be seen, for example, that while initially the I/O processors are more likely to cause a system shutdown, in the long run it is the actuators which are most likely to fail first. If the bound $\leq T$ is omitted from the CSL formula:

- $\mathcal{P}_{=?}[\ \neg \mathit{shutdown}\ \mathcal{U}\ \mathit{fail}_j\ ]$

the model checker computes the long-run failure probability (i.e. as $T \to \infty$). The results are: (1)

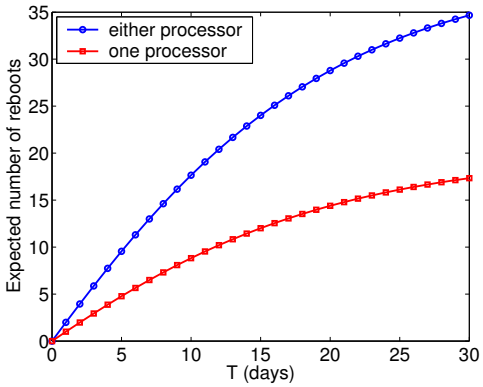
K	Expected time	
	"danger" (hrs)	"up" (days)
1	0.236	14.323
2	0.293	17.660
3	0.318	19.100
4	0.327	19.628
5	0.330	19.809
6	0.331	19.871
7	0.332	19.891
8	0.332	19.897

(a) Expected total time spent in states "danger" and "up" before system shutdown occurs

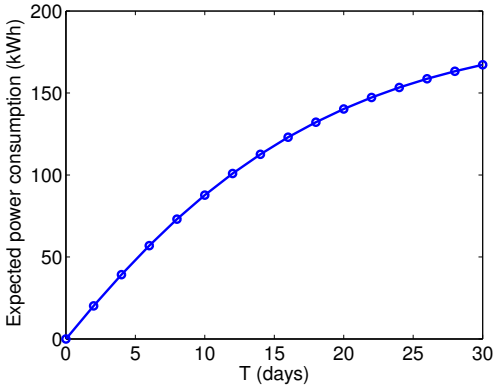


(b) The probability that each failure type is the eventual cause of system shutdown

Fig. 8. System reliability for different values of the parameter K (max. cycles skipped)



(a) Expected number of reboots that occur in the first T days of operation



(b) Expected power consumption over the first T days of operation

Fig. 9. Further measures of system performance over time

0.6216 (2) 0.0877 (3) 0.0484 (4) 0.2423. Note that these sum to one, i.e. the system will eventually shut down with probability 1.

For the second part of the analysis of this case study, the states of the model are classified into three types: "up", where everything is functioning, "shutdown", where the system has shut down, and "danger", where a (possibly transient) failure has occurred but has yet to cause a shutdown (e.g. if the I or O processor has failed but the M processor has yet to detect this). Figure 7 shows the expected time spent by the system in each class of state over a period of T time units. The results are again plotted two ranges of T: the first 24 hours and the first 30 days of system operation.

This analysis was accomplished in PRISM using its support for reward-based properties. For each class of states, a cost structure which assigns a cost of 1 to states in the class and 0 to all others is used. The property:

- $\mathcal{R}_{=?}[\ \mathcal{C}^{\leq T}\ ]$

is then used to compute the expected cost cumulated by the system over T time units. Using the same assignments of costs, the expected total time spent in "up" and "danger" states before the system is shut down can also be analysed. This is done using the property:

- $\mathcal{R}_{=?}[\ \mathcal{F}\ \mathit{shutdown}\ ]$

This demonstrate the use of probabilistic model checking to study the effect of variations in system parameters on its performance. Recall from Section 4 that the control system in the case study features a parameter K, the number of skipped cycles which the main processor will wait before deciding that the input/output processors have failed. Figure 8(a) shows results for the above property over a range of values for K. Observe that increasing the value of K increases the expected time until failure, but also has an adverse

effect on the expected time spent in "danger" states. In a similar fashion, using a property from earlier in this section, Figure 8(b) plots the probability that each of the four types of system failure is the eventual cause of system shutdown for different values of K.

Finally, to illustrate that probabilistic model checking of cost-based properties permits computation of a wide range of performance measures, Figure 9 shows some further results of the analysis of the case study. Figure 9(a) plots the expected number of reboots that occur in the first T days of operation. This is achieved by assigning a cost of 1 to all transitions of the model which correspond to a processor rebooting and 0 to all other states and transitions. Figure 9(b) shows the power consumption of the system over T days. This is done by assigning each state of the model a rate of power consumption based on the components of the system operational in that state. Power consumption rates of 3, 5 and 8 Watts are assumed for each sensor, actuator and processor, respectively.

In the experiments presented in this section, the number of states in the CTMC, which depends on the parameter K, varied between 2,633 and 7,703. The models were all constructed in less than second. For the model checking itself, the time required to compute each value varied from a few seconds to a few minutes, using a standard workstation.

6. CONCLUSION

This paper illustrated how probabilistic model checking, a formal verification technique which has already been applied to a wide range of domains including distributed randomised algorithms, real-time protocols, security protocols and dynamic power management, can be used to analyse dependability properties of controller-based systems. The probabilistic model checker PRISM has a simple system description language which provides an intuitive way to construct complex Markov models. Properties to be checked are specified using temporal logic, which allows reasoning about non-trivial behaviour. The tool also allows use of the efficiency improvements which have been developed in this area. A further advantage of this formal verification approach is that it could easily be combined with more traditional non-probabilistic verification processes, which are becoming increasingly common, particularly in safety-critical areas.

For more detailed information about probabilistic model checking, PRISM and its application to the case study described in this paper, see the tool website: www.cs.bham.ac.uk/~dxp/prism.

7. ACKNOWLEDGEMENTS

This work was partially supported by EPSRC grants GR/S46727 and GR/S11107.

REFERENCES

R. Alur and T. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.

A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In R. Alur and T. Henzinger, editors, *Proc. 8th International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *LNCS*, pages 269–276. Springer, 1996.

C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. On the logical characterisation of performability properties. In U. Montanari, J. Rolim, and E. Welzl, editors, *Proc. 27th International Colloquium on Automata, Languages and Programming (ICALP'00)*, volume 1853 of *LNCS*, pages 780–792. Springer, 2000.

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

C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In J. Baeten and S. Mauw, editors, *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *LNCS*, pages 146–161. Springer, 1999.

R. Bryant. Binary decision diagrams and beyond: Enabling technologies for formal verification. In *Proc. International Conference on Computer-Aided Design (ICCAD'95)*, pages 236–243, 1995.

E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 1999.

L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.

M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker. A formal analysis of Bluetooth device discovery. In *Proc. 1st International Symposium on Leveraging Applications of Formal Methods (ISOLA'04)*, 2004. To appear.

M. Kwiatkowska and G. Norman. Verifying randomized Byzantine agreement. In D. Peled and M. Vardi, editors, *Proc. Formal Techniques for Networked and Distributed Systems (FORTE'02)*, volume 2529 of *LNCS*, pages 194–209. Springer, 2002.

M. Kwiatkowska, G. Norman, and D. Parker. PRISM 2.0: A tool for probabilistic model checking. In *Proc. 1st International Conference on Quantitative Evaluation of Systems*

(QEST'04), pages 322–323. IEEE Computer Society Press, 2004.

M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Performance analysis of probabilistic timed automata using digital clocks. In K. Larsen and P. Niebert, editors, *Proc. Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, volume 2791 of *LNCS*, pages 105–120. Springer-Verlag, 2003a.

M. Kwiatkowska, G. Norman, and R. Segala. Automated verification of a randomized distributed consensus protocol using Cadence SMV and PRISM. In G. Berry, H. Comon, and A. Finkel, editors, *Proc. 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *LNCS*, pages 194–206. Springer, 2001.

M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol. *Special Issue of Formal Aspects of Computing*, 14:295–318, 2003b.

R. Lanotte, A. Maggiolo-Schettini, and A. Troina. Automatic analysis of a non-repudiation protocol. In *Proc. 2nd International Workshop on Quantitative Aspects of Programming Languages (QAPL'04)*, 2004.

J. Muppala, G. Ciardo, and K. Trivedi. Stochastic reward nets for reliability prediction. *Communications in Reliability, Maintainability and Serviceability*, 1(2):9–20, July 1994.

G. Norman, D. Parker, M. Kwiatkowska, and S. Shukla. Evaluating the reliability of NAND multiplexing with PRISM. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2005. To appear.

G. Norman, D. Parker, M. Kwiatkowska, S. Shukla, and R. Gupta. Formal analysis and validation of continuous time Markov chain based system level power management strategies. In W. Rosenstiel, editor, *Proc. 7th Annual IEEE International Workshop on High Level Design Validation and Test (HLDVT'02)*, pages 45–50. IEEE Computer Society Press, 2002.

G. Norman and V. Shmatikov. Analysis of probabilistic contract signing. In A. Abdallah, P. Ryan, and S. Schneider, editors, *Proc. BCS-FACS Formal Aspects of Security (FASec'02)*, volume 2629 of *LNCS*, pages 81–96. Springer, 2003.

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.

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