FACT: A Probabilistic Model Checker for Formal Verification with Confidence Intervals

Radu Calinescu[1(✉)], Kenneth Johnson[2], and Colin Paterson[1]

[1] Department of Computer Science, University of York, York, UK
Radu.Calinescu@york.ac.uk
[2] School of Computer and Mathematical Sciences, Auckland University of Technology, Auckland, New Zealand

Abstract. We introduce FACT, a probabilistic model checker that computes confidence intervals for the evaluated properties of Markov chains with unknown transition probabilities when observations of these transitions are available. FACT is unaffected by the unquantified estimation errors generated by the use of point probability estimates, a common practice that limits the applicability of quantitative verification. As such, FACT can prevent invalid decisions in the construction and analysis of systems, and extends the applicability of quantitative verification to domains in which unknown estimation errors are unacceptable.

1 Introduction

The development of quantitative verification [8,11] over the past fifteen years represents one of the most prominent recent advances in system modelling and analysis. Given a Markov model that captures relevant states of a system and the probabilities or rates of transition between these states, the technique can evaluate key reliability and performance properties of the system. This capability and the emergence of efficient probabilistic model checkers such as PRISM [10] and MRMC [9] have led to adoption in a wide range of applications [14].

Despite the success of quantitative verification, the usefulness of its results depends on the accuracy of the analysed models. Obtaining accurate Markov models is difficult. Although model states and transitions are typically easy to identify (e.g., through static code analysis for software systems), transition probabilities and rates need to be estimated. The common practice is to obtain these estimates through model fitting to log data or run-time observations [4,15], or from domain experts. In either case, the values used in the analysed models contain estimation errors. These errors are then propagated and may be amplified by quantitative verification (since Markov models are nonlinear), producing imprecise results that can lead to invalid design or verification conclusions.

The FACT[1] probabilistic model checker introduced in our paper is not affected by this problem. As described in Sect. 2, FACT can compute confidence intervals for the properties of a common class of parametric (discrete-time) Markov chains for which observations of the transitions associated with

[1] Formal verificAtion with Confidence inTervals.

© Springer-Verlag Berlin Heidelberg 2016
M. Chechik and J.-F. Raskin (Eds.): TACAS 2016, LNCS 9636, pp. 540–546, 2016.
DOI: 10.1007/978-3-662-49674-9_32

the unknown probabilities are available. The operation of FACT (presented in Sect. 3) is underpinned by recent theoretical results from [2], and the tool integrates the PRISM parametric quantitative verification engine (first introduced in version 4.2 of PRISM), the MATLAB convex optimisation toolbox YALMIP [13] and a purpose-built hill climbing optimiser. The modular architecture of the tool (discussed in Sect. 4) makes it easy to replace these components with functionally equivalent ones and to extend the tool. FACT and the models from the case studies summarised in Sect. 5 are available on our project website http://www-users.cs.york.ac.uk/~cap/FACT.

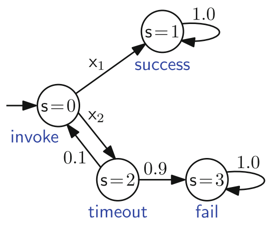
2 Formal Verification with Confidence Intervals

FACT parametric Markov chains (PMCs) are specified in an extended version of the PRISM high-level modelling language [10], which models a system as the parallel composition of a set of *modules*. The state of a *module* is encoded by a set of finite-range local variables, and its state transitions are defined by probabilistic guarded commands that change these variables, and have the general form:

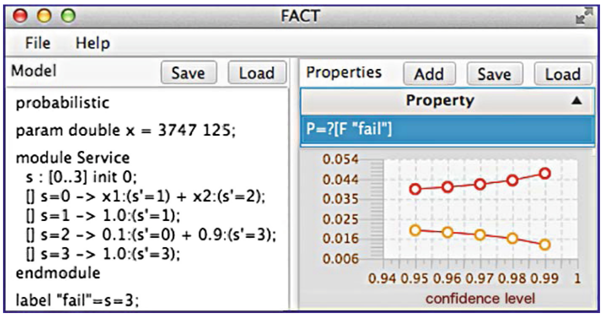
$$[action]\ guard\ {-}{>}\ e_1 : update_1 + e_2 : update_2 + \ldots + e_n : update_n; \qquad (1)$$

In this command, *guard* is a boolean expression over all model variables. If *guard* evaluates to *true*, the arithmetic expression e_i, $1 \le i \le n$, gives the probability with which the $update_i$ change of the module variables occurs. When *action* is present, all modules comprising commands with this *action* have to synchronise (i.e., to carry out one of these commands simultaneously). In a FACT PMC, the expressions $e_1, e_2, \ldots, e_n$ can be unknown (constant) probabilities $\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_n$. These model *parameters* are associated with a declaration:

$$\textsf{\textbf{param double}}\ \mathsf{x} = t_1\ t_2\ \ldots\ t_n; \qquad (2)$$



(a)



```
probabilistic
param double x = 3747 125;
module Service
  s : [0..3] init 0;
  [] s=0 -> x1:(s'=1) + x2:(s'=2);
  [] s=1 -> 1.0:(s'=1);
  [] s=2 -> 0.1:(s'=0) + 0.9:(s'=3);
  [] s=3 -> 1.0:(s'=3);
endmodule
label "fail"=s=3;
```

```
P=?[F "fail"]
```

(b)

Fig. 1. (a) PMC model of a service whose invocations succeed with probability x_1 and time out with probability $\mathsf{x}_2 = 1 - \mathsf{x}_1$, where timed-out invocations are retried with probability 0.1; (b) FACT-generated confidence intervals for the property *'What is the probability that the service cannot be invoked successfully?'* for an instance of the service that was observed completing successfully 3747 times and timing out 125 times.

in which $t_i \in \mathbb{N}$, $1 \le i \le n$, represents the number of transitions associated with $update_i$ that were observed during a period of time when all outgoing transitions from states that satisfy *guard* were monitored and recorded. An example of a simple PMC analysed using FACT is shown in Fig. 1.

FACT PMCs can have multiple sets of parameters (2). For example, the outgoing transitions from state '$\mathsf{s} = 2$' in Fig. 1a could be associated with unknown probabilities pRetry_1 and pRetry_2. The only constraint is that the different sets of parameters (2) are statistically independent. This constraint is satisfied by a broad class of PMCs that includes, for instance, all the models used in the case studies of the PROPhESY tool[2] [5] for analysing parametric Markov chains.

FACT can establish confidence intervals for PMC properties expressed in probabilistic computation tree logic (PCTL) [7] extended with rewards [1]. The current version of FACT supports non-nested probabilistic PCTL properties of the form $\mathcal{P}_{=?}[\Psi]$, where the *path formula* Ψ is defined by the grammar:

$$\begin{aligned} \Psi &::= \mathrm{X}\Phi \mid \Phi \ \mathrm{U} \ \Phi \mid \Phi \ \mathrm{U}^{\le k} \Phi \\ \Phi &::= true \mid a \mid \Phi \wedge \Phi \mid \neg \Phi \end{aligned} \tag{3}$$

with $k \in \mathbb{N}$, a an *atomic proposition* associated with states that satisfy a (e.g., $\mathsf{timeout}$ and $\mathsf{success}$ in Fig. 1a), $p \in [0,1]$, $\bowtie \in \{\ge, >, <, \le\}$, and Φ is a *state formula*. FACT also supports all PCTL reward properties, i.e., the instantaneous, cumulative, reachability and steady-state reward properties defined by:

$$\Phi ::= \mathcal{R}_{=?}[\mathrm{I}^{=k}] \mid \mathcal{R}_{=?}[\mathrm{C}^{\le k}] \mid \mathcal{R}_{=?}[\mathrm{F}\Phi] \mid \mathcal{R}_{=?}[\mathrm{S}]. \tag{4}$$

Defining the semantics of PCTL is beyond the scope of this paper; details are available from [1,7,10].

3 Using FACT

As shown in Fig. 2, FACT users provide a PMC, a PCTL property for analysis, and a range of confidence levels. Given these inputs, the *verification manager* at the core of our tool generates a confidence interval for each confidence level α from the user-specified range in a four-step process. First, *parametric quantitative verification* is used to obtain an algebraic expression for the analysed PCTL property (step 1, executed only once for all confidence levels). This expression, which is recorded in the FACT log, is a rational function of the PMC parameters, e.g., $\frac{9x_2}{10x_1+9x_2}$ for the PCTL property analysed in Fig. 1b. In step 2, *simultaneous confidence intervals* are calculated for each set of parameters (2) containing elements that appear in the algebraic expression from step 1. If there are m such parameter sets, then a confidence level of $\alpha^{1/m}$ is used to calculate the parameter confidence intervals, and these parameter confidence intervals have a "combined confidence level" of $(\alpha^{1/m})^m = \alpha$. Hence, step 3 uses them as input for a *convex optimisation* problem whose solution represents an α confidence interval for the analysed property—a formal proof of this result is available in [2].

[2] http://moves.rwth-aachen.de/research/tools/prophesy/#benchmarks.

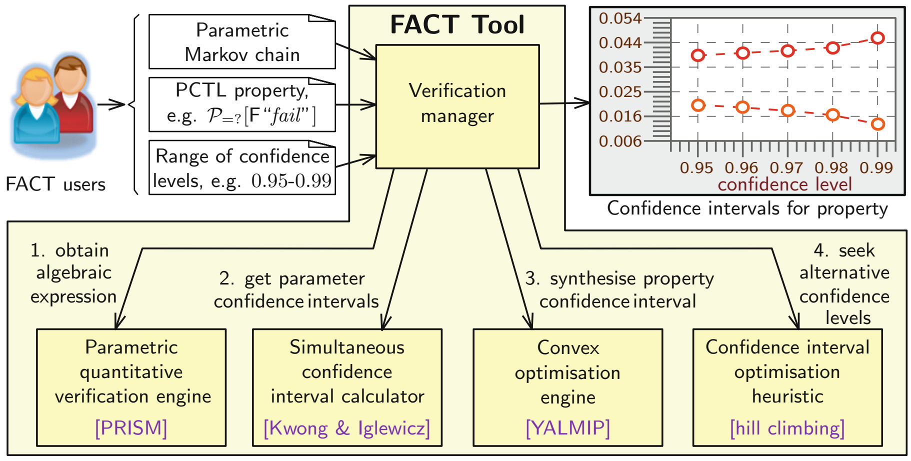

FACT users
Parametric Markov chain
PCTL property, e.g. $\mathcal{P}_{=?}[\mathsf{F}\,\text{"}fail\text{"}]$
Range of confidence levels, e.g. 0.95-0.99
FACT Tool
Verification manager
0.054
0.044
0.035
0.025
0.016
0.006
0.95 0.96 0.97 0.98 0.99
confidence level
Confidence intervals for property
1. obtain algebraic expression
2. get parameter confidence intervals
3. synthesise property confidence interval
4. seek alternative confidence levels
Parametric quantitative verification engine [PRISM]
Simultaneous confidence interval calculator [Kwong & Iglewicz]
Convex optimisation engine [YALMIP]
Confidence interval optimisation heuristic [hill climbing]

Fig. 2. FACT operation and architecture; the technologies used by the current version of the tool (shown in square brackets) can be replaced with alternative technologies

When $m > 1$, using $\alpha^{1/m}$ confidence intervals for each parameter set is unlikely to yield the narrowest possible α confidence interval for the analysed property. For two reasons, using confidence levels $\alpha_i < \alpha^{1/m} < \alpha_j$ for the confidence intervals of parameter sets i and j may produce a narrower α confidence interval:

1. If the number of state-transition observations associated with parameter set j is larger than that for parameter set i, this choice of confidence levels may produce much narrower confidence intervals for parameter set i with an insignificant widening of the confidence intervals for parameter set j;
2. If the analysed property is particularly sensitive to variations in the parameter set i, reducing α_i narrows the confidence intervals for parameter set i and may also narrow the α confidence interval for the analysed property.

Therefore, step 4 uses a *confidence interval optimisation heuristic* to seek alternative confidence levels $\alpha_1, \alpha_2, \ldots, \alpha_m$ such that $\prod_{i=1}^{m} \alpha_i = \alpha$ and using α_i confidence intervals for the i-th parameter set, $1 \leq i \leq m$, produces a narrower α confidence interval for the analysed property. This optimisation can reduce the width of property confidence intervals (e.g., by up to 14 % in the case studies from [2]), but is time consuming since FACT steps 2 and 3 are repeated for each $\alpha_1, \alpha_2, \ldots, \alpha_m$ combination suggested by the heuristic. Hence step 4 is by default switched off in FACT, and the user should switch it on explicitly if needed. There is one typical scenario in which this need arises. This is when FACT is used to verify whether the analysed property is above/below a threshold specified in the system requirements (with some confidence level α), and the threshold falls inside the α confidence interval without the heuristic search. In this scenario, the FACT user should switch on the heuristic search by specifying a non-zero number of search iterations, which may result in a narrower α confidence interval that does not contain the threshold and enables a conclusion to be drawn.

4 Architecture and Implementation

FACT has a modular architecture in which each step of the verification process is carried out by a different module (Fig. 2). We implemented these modules in Java, using the following technologies that can each be easily substituted with alternative technologies (e.g., to extend FACT or to improve its efficiency):

1. The parametric quantitative verification engine is implemented on top of PRISM [10], which it invokes in the background. An alternative implementation based on PARAM [6] is worth exploring.
2. The simultaneous confidence interval calculator implements the (conservative) solution proposed by Kwong and Iglewicz [12], which achieves a good trade-off between computational complexity and precision. Several alternative solutions that deserve investigating are mentioned in [2].
3. The convex optimisation engine uses the MATLAB convex optimisation toolbox YALMIP [13], which it invokes in the background. An implementation based on the non-commercial GNU Octave package (https://www.gnu.org/software/octave/) is worth exploring.
4. The confidence interval optimisation heuristic currently used is hill climbing. Numerous alternative heuristics can be substituted in this module.

5 Case Studies and Experimental Results

To evaluate FACT, we carried out case studies involving the synthesis of confidence intervals for PCTL-encoded reliability, performance and cost properties of

Table 1. Experimental results for the case studies from Sect. 5

PMC	*psets*[a]	*params*[b]	PCTL property	t_{exp}^{c}	t_{CI}^{d}
Web	5	13	$\mathcal{P}_{=?}[\mathrm{F}\ \mathsf{HttpResponse}]$	0.75s	3.96s
			$\mathcal{P}_{=?}[\neg(\mathsf{Database} \vee \mathsf{FileServer})\ \mathrm{U}\ \mathsf{HttpResponse}]$	0.84s	3.43s
			$\mathcal{R}_{=?}^{\mathsf{cost}}[\mathrm{F}\ \mathsf{Done}]$	0.86s	3.31s
			$\mathcal{R}_{=?}^{\mathsf{time}}[\mathrm{F}\ \mathsf{Done}]$	0.89s	3.29s
TAS	3	6	$\mathcal{P}_{=?}[\mathrm{F}\ \mathsf{FailedAlarm}]$	0.24s	4.32s
			$\mathcal{P}_{=?}[\neg\mathsf{Done}\ \mathrm{U}\ \mathsf{FailedService}]$	0.12s	2.82s
			$\mathcal{P}_{=?}[\neg\mathsf{Done}\ \mathrm{U}\ \mathsf{FailedAlarm}\{\mathsf{MedicalAnalysis}\}]$	0.11s	2.78s
LWB	1	2	$\mathcal{R}_{=?}^{\mathsf{power}}[\mathrm{S}]$	0.24s	3.03s
			$\mathcal{R}_{=?}^{\mathsf{energy}}[\mathrm{F}\ \mathsf{StartedUp}]$	0.27s	2.98s
BRP	2	4	$\mathcal{P}_{=?}[\mathrm{F}\ \mathsf{SenderNoSuccessReport}]$	0.44s	31.6s
Z	2	4	$\mathcal{R}_{=?}^{\mathsf{numTests}}[\mathrm{F}\ \mathsf{DecisionMade}]$	0.15s	5.41s

[a]Number of parameter sets (2) in the PMC
[b]Total number of PMC parameters
[c]Time to compute algebraic expression
[d]Time to synthesise confidence interval

parametric Markov chains modelling systems from different application domains. Table 1 summarises the experimental results obtained for the PMCs of:

– a web application taken from [2] (Web);
– a tele-assistance service-based system adapted from [3,4] (TAS);
– the low-power wireless bus communication protocol taken from [2] (LWB);
– the bounded retransmission protocol from the PROPhESY [5] site (BRP);
– the Zeroconf IP address selection protocol from the PARAM [6] website (Z).

The timing results were obtained on a standard OS X 10.8.5 MacBook computer with 1.3 GHz Intel Core i5 processor and 8 GB 1600 MHz DDR3 RAM. The models, PCTL property files, results and descriptions for all case studies are available on our FACT website http://www-users.cs.york.ac.uk/~cap/FACT.

These case studies demonstrated several key benefits of our probabilistic model checker. First, FACT supports the analysis of systems for which state transition probabilities are unknown, but observations of these transitions are available from logs or run-time monitoring. Second, it enables the analysis of reliability, performance and other non-functional properties of systems at the required confidence level. This approach is better aligned with the current industrial practice than traditional quantitative verification. Third, it can prevent invalid design and verification decisions. In many scenarios, the quantitative analysis of Markov models built using point estimates of the unknown transition probabilities misleadingly suggested that requirements were met. In contrast, FACT showed that this was only the case with low confidence levels that are typically deemed unacceptable in practice. Last but not least, our case studies showed that FACT can be used to analyse systems from multiple domains.

References

1. Andova, S., Hermanns, H., Katoen, J.-P.: Discrete-time rewards model-checked. FORMATS 2003. LNCS, vol. 2791, pp. 88–104. Springer, Heidelberg (2003)
2. Calinescu, R., Ghezzi, C., Johnson, K., Pezze, M., Rafiq, Y., Tamburrelli, G.: Formal verification with confidence intervals to establish quality of service properties of software systems. IEEE Trans. Reliab. **PP**(99), 1–16 (2015)
3. Calinescu, R., Johnson, K., Rafiq, Y.: Developing self-verifying service-based systems. In: ASE 2013, pp. 734–737 (2013)
4. Calinescu, R., Rafiq, Y., Johnson, K., Bakir, M.E.: Adaptive model learning for continual verification of non-functional properties. In: ICPE 2014, pp. 87–98 (2014)
5. Dehnert, C., Junges, S., Jansen, N., Corzilius, F., Volk, M., Bruintjes, H., Katoen, J.-P., Ábrahám, E.: PROPhESY: A PRObabilistic ParamEter SYnthesis Tool. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 214–231. Springer, Heidelberg (2015)
6. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: PARAM: a model checker for parametric Markov models. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 660–664. Springer, Heidelberg (2010)
7. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects Comput. **6**(5), 512–535 (1994)

8. Haverkort, B.R., Katoen, J.-P., Larsen, K.G.: Quantitative verification in practice. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010, Part II. LNCS, vol. 6416, pp. 127–127. Springer, Heidelberg (2010)
9. Katoen, J.-P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Perform. Eval. **68**(2), 90–104 (2011)
10. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)
11. Kwiatkowska, M.Z.: Quantitative verification: models, techniques and tools. In: ESEC-FSE 2007, pp. 449–458 (2007)
12. Kwong, K.-S., Iglewicz, B.: On singular multivariate normal distribution and its applications. Comput. Stat. Data Anal. **22**(3), 271–285 (1996)
13. Löfberg, J.: Automatic robust convex programming. Optim. Methods Softw. **27**(1), 115–129 (2012)
14. Norman, G., Parker, D.: Quantitative verification: formal guarantees for timeliness, reliability and performance. Technical report, London Mathematical Society and the Smith Institute for Industrial Mathematics and System Engineering (2014)
15. Su, G., Rosenblum, D.S.: Asymptotic bounds for quantitative verification of perturbed probabilistic systems. In: Groves, L., Sun, J. (eds.) ICFEM 2013. LNCS, vol. 8144, pp. 297–312. Springer, Heidelberg (2013)