Expanding the Boundaries of Performance Requirement Representation with Performance Trees

Tamas Suto

Department of Computing, Imperial College London,
180 Queen's Gate, London SW7 2AZ, United Kingdom

Abstract

We introduce Performance Trees (PTs), a novel representation formalism for the specification of model-based performance queries. Traditionally, stochastic logics have been the prevalent means of performance requirement expression; however, in practice, their use amongst system designers is limited on account of their inherent complexity and restricted expressive power. PTs are a more accessible alternative, in which performance queries are represented by hierarchical tree structures. This allows for the convenient visual composition of complex performance questions, and enables not only the verification of stochastic requirements, but also the direct extraction of performance measures. In addition, the tree formalism offers a superset of the expressiveness of Continuous Stochastic Logic (CSL), since all CSL formulae can be represented as PTs.

PTs can be used to capture passage time, transient, steady-state and higher order queries of varying levels of sophistication. While they are conceptually independent of the underlying stochastic modelling formalism, in many cases the tree operators we use are already backed up by good algorithmic and tool support for both stochastic verification and performance measure extraction.

Key words: Performance Trees, stochastic logics, performance analysis, model checking, measure extraction

1 Introduction

Systems engineers are faced with high expectations to design and build systems that meet end-user operational performance requirements – an especially challenging task for large-scale, high-throughput distributed systems, such as

Email address: `suto@doc.ic.ac.uk` (Tamas Suto).

cluster computers and telecommunication networks. An established pipeline for determining whether a given system meets its expected performance is to:

a. construct a mathematical model of its operation (using some stochastic modelling formalism)
b. express associated performance-related queries in terms of requirements and measures (using a stochastic logic or other methodology)
c. apply specialised stochastic model checking or quantitative analysis software to resolve the queries

Our current work focuses primarily on step (b) of the process, but we also present a brief overview of steps (a) and (c) in order to convey a clearer understanding of its wider context.

1.1 Limitations of Performance Analysis

Many real-life systems exhibit random or probabilistic behaviour, which makes it difficult to predict individual events. However, it is often possible to use probability distributions to characterise and model this behaviour mathematically. Stochastic modelling formalisms encapsulate such systems of distributions in an elegant manner. Various formalisms exist, each of which characterises system behaviour at a different level of abstraction. We distinguish between two levels of models. *Low-level* modelling formalisms, such as Markov processes, semi-Markov processes [1] and generalised semi-Markov processes [2], provide a raw representation of the system in terms of its states and transitions, and in many cases are amenable to numerical analysis. *High-level* models, such as stochastic Petri nets [3] and stochastic process algebras [4, 5] abstract the level of detail required for model specification and avoid the need for the tedious enumeration of every system state and transition. Whilst it is sometimes the case that analysis can be performed on the high-level model directly, these are usually mapped onto low-level models for detailed performance analysis.

Having created a stochastic model of the system, it needs to be decided what performance measures are of interest. For example, it may be part of a service level agreement that 95% of the time, an SMS message should take less than 5 seconds to travel between mobile phone handsets. It is common to capture such requirements in a logical formula, using a language such as CSL. This exploits the strength of logical performance specification, namely the ability to compose performance-related requirements systematically and concisely. Two criticisms of such formalisms might be, however, that they obscure the question being asked and do not provide a complete set of usable performance questions for the modeller. Many performance questions of value to system designers can not be asked, due to the limitation in expressiveness of current stochastic logics [6–13]. Such questions may relate to performance measures that need to be extracted from the model directly in some way. We therefore seek to present a new framework for performance query specification which:

(1) allows the specification of performance queries in a clearer and more accessible way,
(2) provides an enhanced set of performance questions to the modeller by allowing not only performance requirement verification on the model, but also the extraction of quantitative performance measures of interest,
(3) maintains the ability to express performance measures concisely, compositionally and systematically.

2 A Tree Formalism for Query Representation

In the past, various logical formalisms have been designed for the representation of performance requirements. However, even though these logical formalisms are very powerful, they lack the accessibility necessary to attract a larger and more diverse user base, and they are also unable to express certain types of performance requirements. Performance queries can become fairly complex, hence, it is desirable to devise a concise, yet complete representation that summarises the essence of a query in a less esoteric way than stochastic logics do. To cater for this requirement and to overcome the indicated shortcomings, we have developed the *Performance Tree* [14] formalism, a graphical alternative for compositional performance query specification, based on a hierarchical tree structure. We propose a general framework that allows for the expression of a wide range of performance requirements and measures in a uniform manner, regardless of the underlying modelling formalism employed.

A particular instance of a Performance Tree consists of various nodes interconnected by arcs. Nodes in the tree can be of two types. They can either represent operations or values. An *operation node* has inputs and outputs, which are represented by subnodes and supernodes respectively. They symbolise operations that will be performed during the verification and analysis procedure. *Value nodes*, on the other hand, only carry information that form the fundamental building blocks of the performance query, and hence have no subnodes. They symbolise the arguments of the operations. The following syntax describes the operation and the possible arguments (subnodes) of each node.

2.1 Syntax

Operation Nodes:

The **?** operator is the topmost node of a tree. It represents the overall result of the performance query.

The **;** operator is the sequential execution operator, which allows multiple performance requirements and measures to be composed together into one performance query. This operator is especially useful for the identification of optimisation opportunities across several sub-queries. The operator must have at least two subnodes and its result is the list of combined results of the individual sub-queries it combines.

The $\vee\!\!\!\wedge$ operator performs a boolean disjunction or conjunction operation. $\vee\!\!\!\wedge \in \{\vee, \ \wedge\}$. It has two arguments, both of which need to represent a truth value.

The $\neg$ operator performs a boolean negation operation. It has one argument, which must represent a truth value.

The $\bowtie$ operator performs a binary comparison operation. $\bowtie \ \in \{<, \ \leq, \ =, \ \geq, \ >\}$. It has two arguments, both of which have a numerical value, and it returns a truth value.

The $\oplus$ operator represents an arithmetic operation. $\oplus \in \{+, \ -, \ *, \div\}$. It has two arguments, both of which have a numerical value, and it returns a numerical result.

The **PTD** operator represents a passage time density. The arguments (subnodes) define the passage and the node itself represents the density of time for the passage to take place between two sets of states. There must always be at least two sets of states provided (start and target states), but optional constraints relating to actions, states or rewards (represented by the range operator $[\![\ldots]\!]$) can also be supplied.

The **Dist** operator represents a passage time distribution. It takes a passage time density as an argument and converts it into a passage time distribution.

The **Conv** operator represents the convolution of two passage time densities or distributions. The operator takes two arguments, which can both either be densities or distributions, and returns the convoluted function.

The **Moment** operator represents the raw moments of a passage time density function. A single moment generating function can provide us with multiple valuable metrics, since we can derive any number of central moments, the first of which is the expected value, second the variance, *etc.* The operator has two arguments; the first being the rank of the moment that we want to calculate (*e.g.* 3^{rd} moment) and the second being the passage time density, distribution or convolution that we calculate the moments from. The result is a numerical value.

The **SS:P** operator represents the steady-state probability for a given set of states.

The **SS:S** operator represents the set of states that have a steady-state probability of a certain value or within a certain range.

The **FR** operator represents the average firing rate of a certain transition, *i.e.* the average occurrence of a certain action.

The **InInterval** operator determines whether a numerical value is within a certain interval or within multiple intervals. It returns a truth value.

The **InStates** operator is responsible for returning a truth value that expresses whether a certain state or set of states is included in or corresponds to another set of states.

The **ProbInInterval** operator returns the probability with which the passage takes place in a certain amount of time, defined by a time range.

The **ProbInStates** operator corresponds to the transient probability measure of being in a certain set of states at a given instant in time, having started from a particular set of states. The first argument is the set of start states, the second argument is the set of target states and the third argument is the time instant of interest.

The **StatesAtTime** operator returns the set of states that the system can occupy at a given time instant with a certain probability. The first argument represents the time instant and the second the probability value or range.

Value Nodes:

The $[\![\ldots]\!]$ node represents a range / interval. It has two arguments, both of which have a numerical value.

The **States** node represents a set of states. The first annotation identifies the set of states either through state labels or by referencing states directly, and the second annotation is a label representing the type of the states. Labels identify states by specifying conditions on the model. We have two sets of atomic propositions, $SAP = \{\text{state and action labels}\}$ and $TAP = \{\mathit{start}, \mathit{target}, \mathit{incl.}, \mathit{excl.}, \mathit{time}, \mathit{prob.}, \mathit{reward}, \mathit{moment}, \emptyset\}$. We also have $\mathit{State} ::= a \mid tt \mid \mathit{State} \wedge \mathit{State} \mid \neg\mathit{State}$, where $a \in SAP$, and $\mathit{Type} \in TAP$.

The **Actions** node represents a set of actions. The first annotation identifies individual actions either through a set of labels or by referencing them directly. The second represents the type of action. We have $\mathit{Action} ::= a \mid tt \mid \mathit{Action} \wedge \mathit{Action} \mid \neg\mathit{Action}$, where $a \in SAP$ and $\mathit{Type} \in TAP$.

The **Num** node represents a numerical value. The first annotation is the numerical value itself, while the second identifies the type of the numerical value. We have that $Integer \in \mathbb{Z}$, $Real \in \mathbb{R}$ and $Type \in TAP$.

The **Bool** node represents a truth value. Its annotation is the truth value itself. We have that $Boolean \in \{true, \ false\}$.

2.2 Examples

PTs are best appreciated when demonstrated on specific examples. Interesting requirements in terms of steady-state are expressed in Query 1 (*"What is the productivity of the system, defined as the sum of the mean firing rate of action 'processed at A' multiplied by 100, and the mean firing rate of action 'processed at B' multiplied by 200?"*).

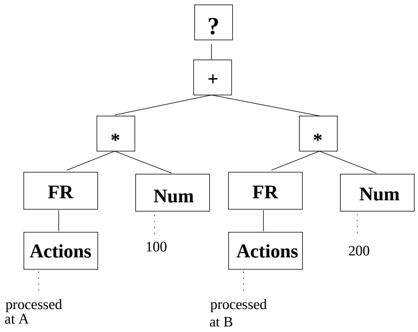

Fig. 1. Query 1

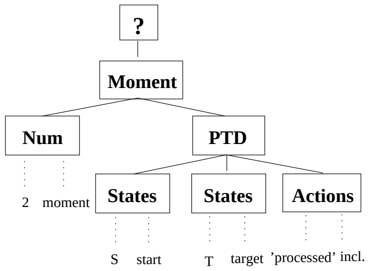

Fig. 2. Query 2

More complex requirements worth describing are to be found in queries 2 and 3. Query 2 addresses the probability of a passage from one set of states to another, given the time it takes to complete the passage and the total accumulated reward along the passage (*"What is the probability of the passage from the set of states S to the set of states T completing in less than* 34 *time units, given that the total reward accumulated along the passage is not less than* 10*?"*). Query 3 is also interested in a probability measure of a passage completing in a given time, and also wants to know the passage time density of the passage (*"What is the probability that a passage from the set of states S to the set of states T will complete in* 50 *time units, and what is the density of time that it takes to complete this passage?"*).

2.3 Performance Trees vs. Stochastic Logics

As demonstrated, the Performance Tree formalism is well-equipped for specifying a large variety of system performance-related queries. The advantage of using Performance Trees as a representation mechanism lies in their broad expressiveness and the fact that they can be applied to general stochastic systems, without the need to rely on any underlying modelling methodology in particular. A further appealing feature is that they can not only be used for the verification of properties on stochastic models, but also for the extraction

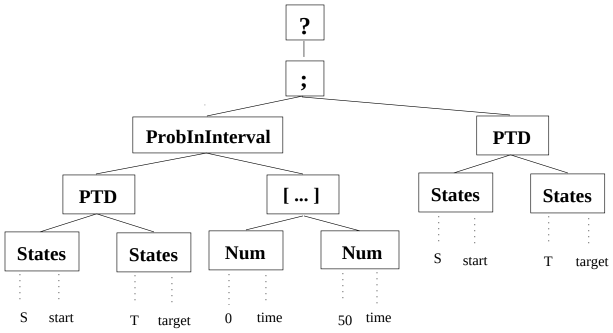

Fig. 3. Query 3

of valuable measures from them (such as obtaining passage time distributions and densities, higher moments, firing rates of actions, *etc.*).

In contrast, stochastic logics have been used successfully for some time now and have subsequently firmly established themselves in the performance community as the representation formalism of choice for stochastic model checking. Even though they enjoy such popularity, we believe that current stochastic logics lack a certain degree of expressiveness that would in many cases be desirable. It is possible to introduce extensions to existing logics or to merge them into new, possibly more powerful formalisms, in order to extend the range of requirements that can be catered for, but we regard Performance Trees as a more powerful and more convenient alternative. In addition, most of the underlying mathematical techniques and tools necessary for evaluating queries expressed as Performance Trees already exist. Also, due to the ability to depict performance query representations graphically, in a way that reflects the logical structure of natural language performance queries, Performance Trees can be understood and used perhaps more naturally than stochastic logic formulae.

The Performance Tree formalism subsumes CSL, since it is capable of expressing everything that CSL can. However, to emphasise the differences in terms of expressiveness between the two formalisms, we give a brief list of the types of requirements that stochastic logics are unable to represent, but which Performance Trees were designed to express: distributions, densities and convolutions, higher moments, firing rates of transitions, arithmetic operations, included states and excluded actions along a passage and multiple probability and time intervals.

References

[1] R. Pyke, "Markov renewal processes: Definitions and preliminary properties," *Annals of Mathematical Statistics*, vol. 32, pp. 1231–1242, December 1961.

[2] J. Matthes, "Zur Theorie der Bedienungsprozesse," in *Transactions of the 3rd Prague Conference on Information Theory, Statistical Decision Functions and Random Processes*, pp. 513–528, 1962.

[3] M. Ajmone Marsan, G. Conte, and G. Balbo, "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems," *ACM Transactions on Computer Systems*, vol. 2, pp. 93–122, May 1984.

[4] J. Hillston, *A Compositional Approach to Performance Modelling*, vol. 12 of *Distinguished Dissertations in Computer Science*. Cambridge University Press, 1996. ISBN 0 521 57189 8.

[5] H. Hermanns, *Interactive Markov Chains*. PhD thesis, Universität Erlangen-Nürnberg, July 1998.

[6] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Verifying continuous-time Markov chains," in *Computer-Aided Verification*, vol. 1102 of *Lecture Notes in Computer Science*, pp. 269–276, Springer-Verlag, 1996.

[7] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Model checking continuous-time Markov chains," *ACM Transactions on Computational Logic*, vol. 1, no. 1, pp. 162–170, 2000.

[8] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen, "On the logical characterisations of performability properties," in *Proceedings of ICALP 2000*, vol. 1853 of *Lecture Notes in Computer Science*, pp. 780–792, Springer-Verlag, 2000.

[9] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen, "Model-checking algorithms for continuous-time Markov chains," *IEEE Transactions on Software Engineering*, vol. 29, pp. 524–541, June 2003.

[10] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle, "Towards model checking stochastic process algebra," in *IFM 2000, Proceedings of 2nd International Conference on Integrated Formal Methods*, pp. 420–439, November 2000.

[11] C. Baier, L. Cloth, B. R. Haverkort, M. Kuntz, and M. Siegle, "Model checking action- and state-labelled Markov chains," *DSN'04, Proceedings of International Conference on Dependable Systems and Networks*, pp. 701–710, June 2004.

[12] B. R. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen, and C. Baier, "Model checking performability properties," in *DSN'02, Proceedings of International Conference on Dependable Systems and Networks*, pp. 103–112, 2002.

[13] J. T. Bradley, N. J. Dingle, W. J. Knottenbelt, and P. G. Harrison, "Performance queries on semi-Markov stochastic Petri nets with an extended Continuous Stochastic Logic," in *PNPM'03, Proceedings of Petri Nets and Performance Models* (G. Ciardo and W. Sanders, eds.), (University of Illinois at Urbana-Champaign), pp. 62–71, IEEE Computer Society, September 2003.

[14] T. Suto, J. T. Bradley, and W. J. Knottenbelt, "Performance Trees: A new approach to quantitative performance specification," in *MASCOTS'06, Proc.* 14^{th} *IEEE/ACM Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, IEEE Computer Society, 2006. To appear.