Performance Trees: Expressiveness and Quantitative Semantics

Tamas Suto   Jeremy T. Bradley   William J. Knottenbelt

Department of Computing, Imperial College London
180 Queen's Gate, London SW7 2BZ, United Kingdom.
{suto,jb,wjk}@doc.ic.ac.uk

***Abstract*—Performance Trees are a recently-proposed mechanism for the specification of performance properties and measures. They represent an attractive alternative to stochastic logics, since they support traditional stochastic model checking queries, while also allowing for the direct extraction of a wide range of quantitative measures. In this paper we illustrate differences in expressiveness between Performance Trees and Continuous Stochastic Logic (CSL), and present quantitative semantics showing the mathematical basis underlying Performance Tree operators. As a running example, we demonstrate performance query specification with Performance Trees on a stochastic Petri net model of a healthcare system.**

I. Introduction

Many systems – especially those in domains such as telecommunications, finance, healthcare, logistics and defence – perform functions of vital importance. A thorough understanding of the performance characteristics of such systems is necessary in order to ensure that they satisfy their performance-related Quality of Service (QoS) requirements. One way to achieve this understanding is to construct and analyse stochastic performance models, which for reasons of analytical and numerical tractability are usually based on Markov and semi-Markov chains [1], [2]. Because specifying every state and transition in such models of real-life systems is infeasible, various high-level formalisms that can be mapped automatically onto underlying (semi-)Markov chains, such as stochastic Petri nets [2], [3], queueing networks [2], [4], [5] and stochastic process algebras [6]–[10], are used.

After constructing a model, performance characteristics can be extracted through analysis. Such characteristics are normally expressed as performance queries of two kinds: performance measures and performance requirements. Performance measures require quantitative results, e.g. *"In a hospital waiting room, what is the steady-state distribution of the number of patients waiting to be treated?"* Performance requirements, on the other hand, seek a truth value as an answer, e.g. *"In a mobile communications network, does the time taken to send an SMS message between two handsets take less than 5 seconds with at least 95% probability?"*

So far, no single formalism has been developed that enables the concise expression of performance measures *and* performance requirements in a single query. Performance measures are traditionally specified in the tool-specific languages of quantitative analysers such as *SPNP* [11], *Möbius* [12], *DNAmaca* [13], *HYDRA* [14], *SHARPE* [15] and the *APNN-toolbox* [16]. Performance requirements have traditionally been expressed as stochastic logic formulae and are evaluated by model checkers such as *PRISM* [17], *ETMCC* [18] and *MRMC* [19]. Popular stochastic logics include CSL [20]–[23], aCSL [24], asCSL [25], CSRL [22], [26] and eCSL [27]. The strength of using a logical representation is that it enables the concise and rigorous specification of performance requirements and supports elegant query composition and systematic verification. However, stochastic logics are unable to represent many questions of interest to system designers, mainly due to limitations of expressiveness. Further, logical paradigms may seem esoteric to many industrial users. In particular, an expert understanding is required to translate performance requirements expressed in natural language into logical formulae.

Performance Trees [28], a novel approach to performance query representation, provide a standard unifying framework for expressing performance measures and performance requirements. They offer a superset of the expressive power of currently available stochastic logics through a rich selection of concepts and operators familiar to users with an engineering background. These include steady-state and passage time distributions and densities, their moments, transition firing rates, convolutions and arithmetic operations. This power is provided without sacrificing computational tractability, since all operators either impose a trivial computational burden or – at least in the context of (semi-)Markov models – are backed up by known numerical algorithms that are amenable to scalable parallel implementation. Performance Trees have a number of further advantages: they support elegant query composition, they are easily visualised as hierarchical tree structures, and they are applicable in the context of a wide range of modelling formalisms, including stochastic Petri nets, queueing networks and stochastic process algebras, due to the use of an abstract state specification mechanism. Furthermore, it is possible to extend the formalism, either in terms of existing operators by using a parameterised macro mechanism, or by defining new operators.

The purpose of this paper is to illustrate differences in expressiveness between Performance Trees and CSL, to demonstrate the use of Performance Trees in the context of case study examples and to present the formal quantitative semantics underlying Performance Tree operators. We proceed as follows: firstly, we present an overview of CSL in Section II to highlight the extent to which the logic is capable of expressing performance concepts and to provide a basis for comparison with Performance Trees, which are introduced in Section III. To familiarise the reader with the new formalism, several examples of its use in the context of a healthcare model are presented in the case study of Section IV. Using the examples of the case study, we contrast aspects of the applicability and expressive power of CSL with Performance Trees, showing that Performance Trees enable the representation of queries that cannot be expressed in CSL. Section V introduces quantitative semantics for the most important Performance Tree operators to describe their underlying mathematical meaning. A summary of the paper's contributions is given in Section VI, and semantics for some remaining Performance Tree operators are provided in the Appendix.

II. CSL

For our purposes, a semi-Markov CSL (similar to [29]) is defined over a semi-Markov state space, $(S, P, H, \mathcal{L})$, where S is the set of states, P is the embedded probability transition matrix, H is the state holding time distribution matrix and $\mathcal{L}$ is a state labelling function. This labelling function attaches multiple labels to each state, thereby allowing states to be grouped and classified in a convenient way. A general CSL formula is defined as follows [23]:

$$\begin{array}{lcl} \sigma & \stackrel{\text{def}}{=} & tt \mid a \mid \neg\sigma \mid \sigma \wedge \sigma \mid \mathcal{S}_{\boldsymbol{\rho}}(\sigma) \mid \mathcal{P}_{\boldsymbol{\rho}}(\varphi) \\ \varphi & \stackrel{\text{def}}{=} & \mathcal{X}\sigma \mid \sigma\, \mathcal{U}^{\boldsymbol{\tau}} \sigma \end{array}$$

$\mathcal{S}$ represents a steady-state condition and $\mathcal{P}$ represents a passage time condition on a set of paths defined by φ. The values $\boldsymbol{\rho}$ and $\boldsymbol{\tau}$ represent ranges of allowed probabilities and times respectively. The semantics of the logic are expressed by stating the conditions under which a single state s satisfies each clause of a σ-formula; this is expressed by the satisfiability relation $s \models \sigma$. The clause a is a label and a state s satisfies that label if $a \in \mathcal{L}(s)$. Thus, using the negation and conjunction clauses in combination with labelling allows whole sets of states to be defined with a σ-formula. The set of states specified in this manner is written $\mathrm{Sat}(\sigma) = \{s \in S \mid s \models \sigma\}$. The steady-state clause $\mathcal{S}_{\boldsymbol{\rho}}(\sigma)$ defines a set of states $S_1 = \mathrm{Sat}(\sigma)$ and is true if the sum of the steady-state probabilities of the states in S_1 lies in range $\boldsymbol{\rho}$.

The formal semantics of CSL [23] are:

$$\begin{array}{lcll} s & \models & tt & \text{for all } s \\ s & \models & a & \text{iff } a \in \mathcal{L}(s) \\ s & \models & \neg\sigma & \text{iff } s \not\models \sigma \\ s & \models & \sigma_1 \wedge \sigma_2 & \text{iff } s \models \sigma_1 \wedge s \models \sigma_2 \\ s & \models & \mathcal{S}_{\boldsymbol{\rho}}(\sigma) & \text{iff } \Pi_J \in \boldsymbol{\rho} \quad \text{where } J = \mathrm{Sat}(\sigma) \\ s & \models & \mathcal{P}_{\boldsymbol{\rho}}(\varphi) & \text{iff } \mathbb{P}(\sigma \in \mathrm{Path}(s) \mid \sigma \models \varphi) \in \boldsymbol{\rho} \end{array}$$

where Π_J is the steady-state probability of being in any of the states in J, $\mathbb{P}$ denotes a probability and $\mathrm{Path}(s)$ is the set of all paths starting from s. Further, a path ψ satisfies a path formula, φ, as follows:

$$\begin{array}{lcll} \psi & \models & \mathcal{X}\sigma & \text{iff } \exists \psi[1] \models \sigma \\ \psi & \models & \sigma_1\, \mathcal{U}^{\boldsymbol{\tau}} \sigma_2 & \text{iff } \exists t \in \boldsymbol{\tau}\ .(\psi@t \models \sigma_2 \wedge \\ & & & \qquad \forall t' < t, \psi@t' \models \sigma_1) \end{array}$$

where $\psi[1]$ is a state immediately succeeding the start state of ψ; $\psi@t$ is the state that the system is in at time t on the path ψ. The $\mathcal{X}$ path operator is often referred to as the *next state* operator and asserts that the next transition will be made to a σ state. The *time-bounded until* formula $\sigma_1\, \mathcal{U}^{\boldsymbol{\tau}} \sigma_2$ asserts that σ_2 is satisfied at some time instant in the interval $\boldsymbol{\tau}$ and that at all preceding time instants σ_1 holds.

III. Performance Trees

A visualised instance of a Performance Tree consists of a set of nodes, interconnected by arcs, that form a hierarchical tree structure (see examples in Section IV). Nodes in the tree can be of two kinds: operation or value nodes. *Operation nodes* are functions, taking zero or more sub-nodes as arguments and returning a result (e.g. a passage time density). Sub-nodes can be value nodes (usually representing numerical values, states or actions) or operation nodes that return a value of an appropriate type. Table I provides a summary of the currently available Performance Tree operation and value nodes.

We note that parse trees can be used to visualise CSL queries; however, these are generally used as internal tool representations and user-level CSL query specification is still predominantly text-based. By contrast, Performance Trees were designed from the outset to be a graphical user-level specification formalism, although they can also be expressed in a textual form.

A performance query that is expressible as a CSL formula is usually analysed by a model-checker that returns a truth value, indicating whether the model satisfies the requirements encoded in the formula. By contrast, queries expressed as Performance Trees are evaluated by a query interpreter, which can return various kinds – or even a sequence – of results, depending on which operation nodes are used.

The Performance Tree formalism is extensible in two ways. Firstly, whenever new performance operators are necessary to express new types of queries, the set of basic operators can be extended. Naturally, a query interpreter would need to be modified to support any new operators. Secondly, Performance Trees support parameterised macros of complex expressions built from existing operators. In this case, no query interpreter modifications are necessary. An example of this mechanism is used in is the Performance Tree of Query 3 below.

IV. An Illustrative Case Study

Consider the simplified stochastic Petri net model of a healthcare system of Figure 1. Here, there is an initial group of healthy people who fall ill and go to a hospital – arriving either by themselves as walk-in patients or by ambulance. Walk-in patients wait in the waiting room for assessment until a nurse becomes available, while ambulance patients wait on a trolley to be assessed by a nurse with greater urgency. Patients are subsequently either seen by a doctor for treatment, sent for lab tests, or sent for surgery. Once a patient has been discharged from the hospital, (s)he is (optimistically) assumed to be healthy again.

The model is parameterised by the values of P, N and D, which denote the number of tokens on the places *healthy* (people), *nurses* and *doctors*, respectively. To construct performance queries relating to individual patients, we employ the well-known tagged customer concept, which in the Petri net context of the given model means tracking the flow of a token that represents a particular patient's progress through the system. This necessitates the introduction of an extra transition for each existing transition in the net to differentiate between the cases of forwarding tagged and untagged customers. To specify queries involving a tagged patient, we introduce the notation *patient@place*, which is an atomic proposition that is attributed to a state if the token representing the tagged patient is at place *place* in the model.

The first two examples below illustrate performance queries on this model that can be expressed as both CSL formulae and Performance Trees. Two further examples provide performance queries which cannot be represented as logical formulae, due to limitations of CSL's expressiveness.

Query 1: *Is the time from the first person falling ill to the time of discharge from the hospital less than 4 hours (240 minutes) at least 98% of the time?*

This query represents a performance requirement on a passage time quantile involving the transit of a tagged patient. For both CSL and Performance Tree representations of this query, it is convenient to use state labels to identify the source and destination states as follows:

$\textit{fallen ill} := (\#(\textit{healthy}) = P - 1) \wedge (\#(\textit{ill}) = 1) \wedge$
$(\#(\textit{nurses}) = N) \wedge (\#(\textit{doctors}) = D)$
$\textit{in hospital} := (\textit{patient@waiting room}) \vee (\textit{patient@trolley}) \vee$
$(\textit{patient@patient being assessed}) \vee$
$(\textit{patient@ambulance patient being assessed}) \vee$
$(\textit{patient@waiting to be treated}) \vee$
$(\textit{patient@treated by doctor}) \vee (\textit{patient@surgery done}) \vee$
$(\textit{patient@patient recovered}) \vee (\textit{patient@tests done})$
$\textit{discharged} := (\textit{patient@healthy})$

CSL: $s_{\textit{fallen ill}} \models$
$\mathcal{P}_{\geq 0.98}\big((\textit{in hospital})\ \mathcal{U}^{[0,240]}\,(\textit{discharged})\big)$

PT: $?\Big(\textit{InInterval}\Big(\textit{ProbInInterval}\big(\textit{PTD}($
$\textit{States}(\textit{fallen ill,start}),\ \textit{States}(\textit{discharged,target})),$
$[\![\ldots]\!](\textit{Num}(\textit{0,time}),\ \textit{Num}(\textit{240,time}))\big),$
$[\![\ldots]\!]\big(\textit{Num}(\textit{0.98,prob}),\textit{Num}(\textit{1,prob})\big)\Big)\Big)$

In the CSL expression, $s_{\textit{fallen ill}}$ indicates the single state that corresponds to the state label *fallen ill*. It is important to note that in the case of multiple start states, CSL does state-by-state verification of passage time constraints, while Performance Trees verify the constraints using weighted averages over groups of states. To obtain weighted average semantics in CSL for multiple start states, it is necessary to (manually) insert additional states into the model's underlying Markov chain.

The Performance Tree of Figure 2 represents the above query. The *PTD* operator is used to calculate a passage time density. The set of start states for this passage consists of the single state in which the first patient has fallen ill, and the set of target states consists of the states where this patient has been discharged from the hospital. To find the probability with which the passage takes place in under 240 minutes, we use the *ProbInInterval* node to integrate the passage time density over the time range specified by the $[\![\ldots]\!]$ node. To establish whether this probability is at least 0.98, we use the *InInterval* node, which returns a boolean result.

Query 2: *Is the probability of having less than 3 patients in recovery after surgery at time* 120 *greater than* 0.7*?*

Relevant state labels for this query are:

$\textit{initial} := (\#(\textit{healthy}) = P) \wedge (\#(\textit{nurses}) = N) \wedge$
$(\#(\textit{doctors}) = D)$
$<\textit{3 patients recovering} := (\#(\textit{surgery done}) < 3)$

where $\#(p)$ returns the number of tokens on place p in the model and boolean conditions such as $(\#(\textit{surgery done}) < 3)$ are conditions on the marking of the Petri net used, in this case, to label individual states of the state space with the atomic proposition ($<$ *3 patients recovering*).

CSL: $s_{\textit{initial}} \models$
$\mathcal{P}_{>0.7}\big((\textit{true})\ \mathcal{U}^{[120,120]}(<\textit{3 patients recovering})\big)$

PT: $?\Big(\textit{InInterval}\big(\textit{ProbInStates}(\textit{States}(\textit{initial,start}),$
$\textit{States}(<\textit{3 patients recovering,target}),\ \textit{Num}(\textit{120,time})),$
$[\![\ldots]\!](\textit{Num}(\textit{0.7,prob}),\textit{Num}(\textit{1,prob}))\big)\Big)$

In the CSL expression, $s_{\textit{initial}}$ indicates the single state that corresponds to the state label *initial*. Figure 3 shows the corresponding Performance Tree. Since we are representing a transient state query, we use the *ProbInStates* operator, which returns the probability of the system being in a given set of states at a given time instant after some initial marking. To verify whether this probability lies in the interval [0.7,1], we use the *InInterval* operator.

Query 3: *What is the coefficient of variation (the ratio of the standard deviation to the mean) of the time for a patient to be seen, treated and discharged from the hospital?*

CSL: This question cannot be expressed in CSL, since it does not provide the means to extract measures, especially not complex ones involving higher moments of passage time.

PT: $?\Big(\textit{Cov}\big(\textit{PTD}(\textit{States}(\textit{patient arrived,start}),$
$\textit{States}(\textit{discharged,target}))\big)\Big)$
$\textit{Cov}(X) \overset{\textit{def}}{=} /\ \Big(\ \hat{}\ \big(-(\textit{Moment}(\textit{Num}(2,\ \textit{moment}),X),$
$\hat{}\ (\textit{Moment}(\textit{Num}(\textit{1,moment}),\ X),\ \textit{Num}(\ \textit{2,power}))\big),$
$\textit{Num}(\textit{0.5,power})\big),\ \textit{Moment}\big(\textit{Num}(\textit{1,moment}),\ X\big)\Big)$

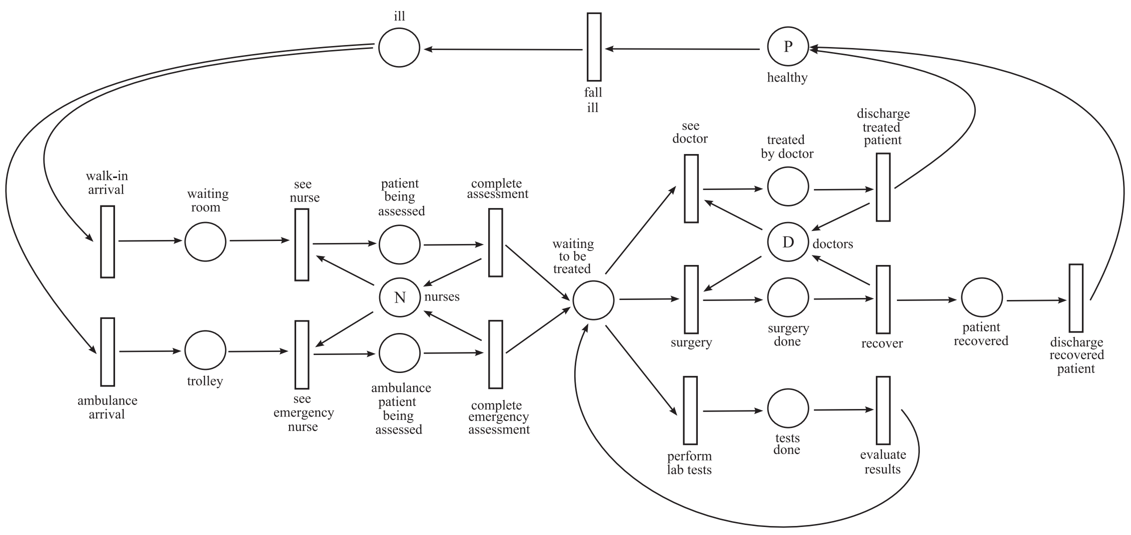

Fig. 1. Patient flow in a hospital environment

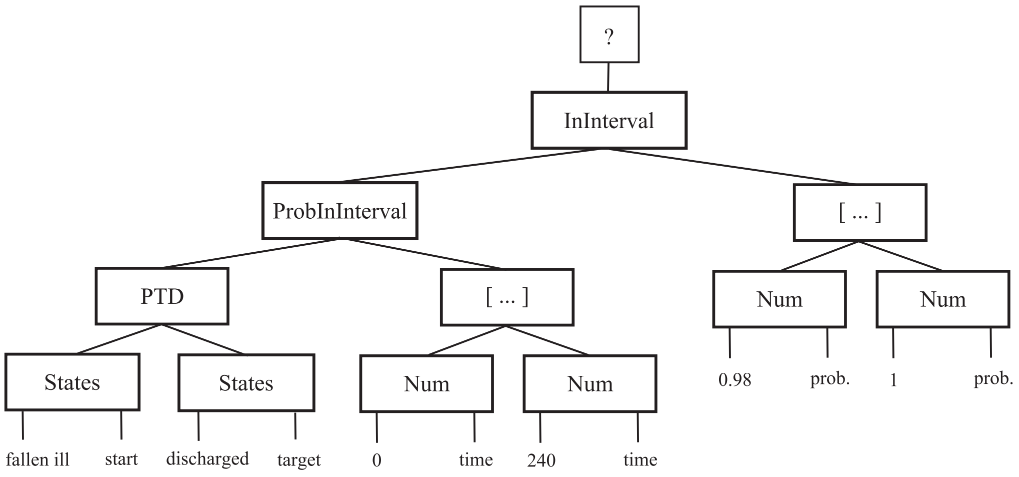

Fig. 2. Performance Tree showing an example of a passage time quantile calculation

The Performance Tree of Figure 4 includes an example of macro expansion, since the operator *Cov* is not part of the standard set of operators.

Query 4: *What is the average rate of occurrence of surgeries, and what is the steady-state probability distribution of the number of patients waiting for treatment and of the number of patients inside and outside of the hospital?*

Relevant state labels for this query are:

$$\begin{aligned}&\mathit{all} := \mathit{true}\\ &\#(\mathit{in\ hospital}) := P - \#(\mathit{outside\ hospital})\\ &\#(\mathit{outside\ hospital}) := \#(\mathit{healthy}) + \#(\mathit{ill})\end{aligned}$$

CSL: A query of this type cannot be expressed in CSL, since it is not capable of calculating the average rate of occurrence of actions or steady-state probability distributions.

PT: $?\Big(;\Big(\mathit{FR}\big(\mathit{Actions}(\mathit{surgery})\big),$
$\mathit{SS{:}P}\big(\mathit{States}(\mathit{all}),\mathit{StateFunc}(\#(\mathit{waiting\ to\ be\ treated}))\big),$
$\mathit{SS{:}P}\big(\mathit{States}(\mathit{all}),\mathit{StateFunc}(\#(\mathit{in\ hospital}))\big),$
$\mathit{SS{:}P}\big(\mathit{States}(\mathit{all}),\mathit{StateFunc}(\#(\mathit{outside\ hospital}))\big)\Big)\Big)$

The Performance Tree for this query is shown in Figure 5. The query consists of a number of independent sub-queries, so we use the *;* operator to combine them. The first part of the query is seeking the average rate of occurrence of an action, which we represent by the firing rate (*FR*) node. The remaining parts of the query address steady-state probability distributions, so the *SS:P* node is used.

V. Quantitative Semantics for Performance Trees

This section presents the formal mathematical basis underlying the most interesting Performance Tree operators. These are presented in the context of (semi-)Markov models and many of them are represented in terms of the Laplace transforms of the quantities sought (see Section V-B1). Note that this section is not intended as a guide to implementors seeking efficient and/or scalable algorithms. These can be found in references such as [13], [14], [30]–[35].

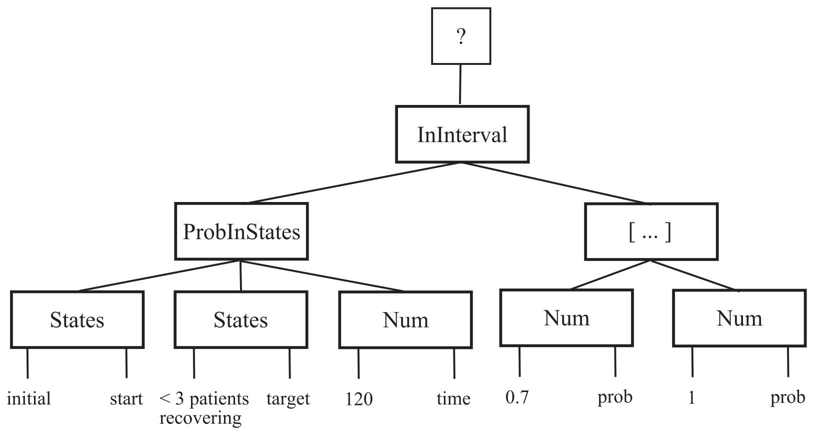

Fig. 3. Performance Tree showing an example of a transient state query

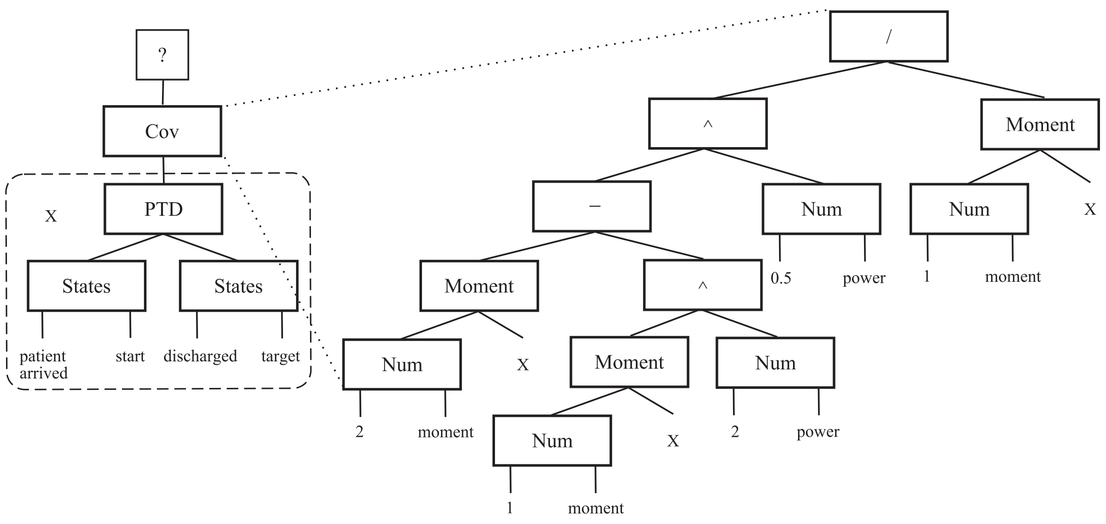

Fig. 4. An example of Performance Tree macro expansion used to calculate the coefficient of variation

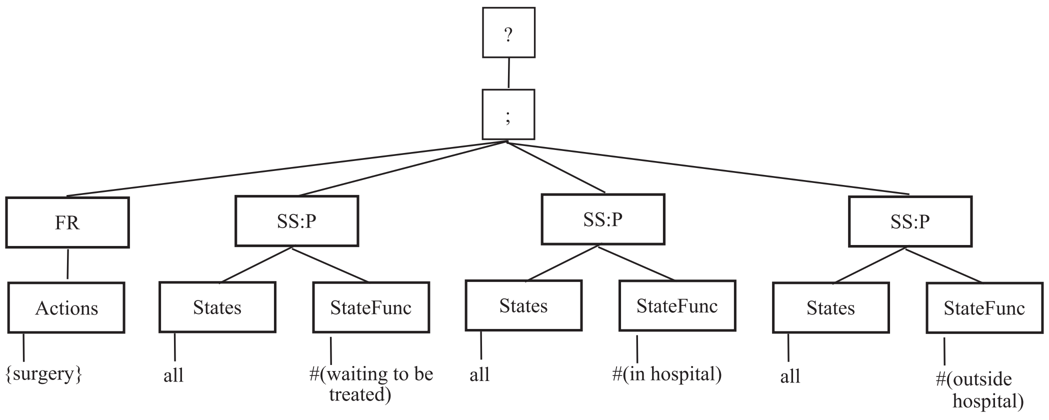

Fig. 5. An example of steady-state measure specification in the healthcare system

A. Notational Conventions

Throughout this section, we adhere to the following notational conventions. Domains used in the description of Performance Tree operators are $\mathcal{S}$, the (finite) set of all states in the model, $\mathcal{A}$, the set of all actions in the model, and $\mathcal{B} = \{true, false\}$. $\mathcal{L} : \mathcal{S} \rightarrow 2^{AP}$ is a labelling function that assigns to a state a label from AP, the set of atomic propositions. $Ivl(I)$ is a function that converts the interval I into the range representation of Performance Trees, such that if $I = [x, y]$ then

$Ivl(I) = [\ldots](Num(x), Num(y))$. Scalar values are denoted by lowercase letters, sets by uppercase letters and compound expressions by E. A $\cdot$ in place of an attribute of a Performance Tree node indicates that the attribute is of no importance in the current context and can assume an arbitrary value.

B. An Introduction to Semi-Markov Performance Models

Consider a Markov renewal process $\{(X_n, T_n) : n \geq 0\}$ where T_n is the time of the nth transition ($T_0 = 0$) and $X_n \in \mathcal{S}$ is the state immediately after the nth transition. Let the kernel of this process be:

$$R(n, i, j, t) = \mathbb{P}(X_{n+1} = j, T_{n+1} - T_n \leq t \mid X_n = i) \quad (1)$$

for $i, j \in \mathcal{S}$. The continuous-time semi-Markov process (SMP), $\{Z(t), t \geq 0\}$, defined by the kernel R, is related to the Markov renewal process by:

$$Z(t) = X_{N(t)} \quad (2)$$

where $N(t) = \max\{n : T_n \leq t\}$ is the number of state transitions that have taken place by time t. Thus $Z(t)$ represents the state of the system at time t. We consider time-homogeneous SMPs, in which $R(n, i, j, t)$ is independent of any previous state, except the last. Thus R becomes independent of n and for any $n \geq 0$ we have:

$$\begin{aligned} R(i, j, t) &= \mathbb{P}(X_{n+1} = j, T_{n+1} - T_n \leq t \mid X_n = i) \\ &= p_{ij} H_{ij}(t) \end{aligned} \quad (3)$$

where $p_{ij} = \mathbb{P}(X_{n+1} = j \mid X_n = i)$ is the state transition probability between states i and j and $H_{ij}(t) = \mathbb{P}(T_{n+1} - T_n \leq t \mid X_{n+1} = j, X_n = i)$, is the sojourn time distribution in state i when the next state is j.

1) Laplace transforms: The Laplace transform is an operation involving an integral transform from real-valued t-space into complex-valued s-space (within which the function may be much more easily manipulated in certain contexts). The Laplace transform $f^*(s)$ of a function $f(t)$ is:

$$f^*(s) = \int_0^\infty e^{-st} f(t)\,\mathrm{d}t \quad (4)$$

where s is a complex number. A useful property of the Laplace transform is that the convolution operation in the time domain is represented by the product in the Laplace domain. A further appealing property is that a cumulative distribution function can be obtained by dividing the Laplace transform of the corresponding probability density function by s. Also, higher moments are easily derived; thus if $f(t)$ is a probability density function of a continuous random variable X, then the nth moment of X is given by:

$$\mathbb{E}(X^n) = (-1)^n f^{*(n)}(0) \quad (5)$$

2) First Passage Times: Consider a finite, irreducible, continuous-time semi-Markov process with N_s states $\{1, 2, \ldots, N_s\}$. Recalling that $Z(t)$ denotes the state of the SMP at time $t \geq 0$, the first passage time [32] from a source state i at time t into a non-empty set of target states J is:

$$P_{iJ}(t) = \inf\{u > 0 : Z(t{+}u) \in J, N(t{+}u) > N(t), Z(t) = i\} \quad (6)$$

For a stationary time-homogeneous SMP, $P_{iJ}(t)$ is independent of t and we have:

$$P_{iJ} = \inf\{u > 0 : Z(u) \in J, N(u) > 0, Z(0) = i\} \quad (7)$$

P_{iJ} has an associated probability density function $f_{iJ}(t)$:

$$\mathbb{P}(t_1 < P_{iJ} < t_2) = \int_{t_1}^{t_2} f_{iJ}(t)\,\mathrm{d}t \quad (8)$$

In general, the Laplace transform of f_{iJ}, $L_{iJ}(s)$, can be computed by solving a set of N_s linear equations:

$$L_{iJ}(s) = \sum_{k \notin J} r^*_{ik}(s) L_{kJ}(s) + \sum_{k \in J} r^*_{ik}(s) \quad (9)$$

where $1 \leq i \leq N_s$ and $r^*_{ik}(s)$ is the Laplace-Stieltjes transform (LST) of $R(i, k, t)$ defined by:

$$r^*_{ik}(s) = \int_0^\infty e^{-st}\,\mathrm{d}R(i, k, t) \quad (10)$$

In case of multiple source states, denoted by set I, the Laplace transform of the passage time density at steady-state is:

$$L_{IJ}(s) = \sum_{k \in I} \alpha_k L_{kJ}(s) \quad (11)$$

where the weight α_k is the probability at equilibrium that the system is in state $k \in I$ at the starting instant of the passage. If π denotes the steady-state vector of the embedded DTMC with one-step transition probability matrix $P = [p_{ij}, 1 \leq i, j \leq N_s]$, then α_k is given by:

$$\alpha_k = \begin{cases} \pi_k / \sum_{j \in I} \pi_j & \text{if } k \in I \\ 0 & \text{otherwise} \end{cases} \quad (12)$$

3) Transient Distributions: Another key modelling result is the transient state distribution, $\pi_{ij}(t)$, of a stochastic process:

$$\pi_{iJ}(t) = \mathbb{P}(Z(t) \in J \mid Z(0) = i) \quad (13)$$

From [36], the Laplace transform of π_{iJ} is:

$$\pi^*_{iJ}(s) = I_{i \in J} \overline{F}^*_i(s) + \sum_{k=1}^{N_s} r^*_{ik}(s) \pi^*_{kJ}(s) \quad (14)$$

where $\overline{F}^*_i(s) = \frac{1}{s}(1 - h^*_i(s))$ is the Laplace transform of the reliability function and $h^*_i(s) = \sum_j r^*_{ij}(s)$ is the LST of the sojourn time distribution in state i. Again for multiple source states, with initial distribution α, we have:

$$\pi^*_{IJ}(s) = \sum_{k \in I} \alpha_k \pi^*_{kJ}(s) \quad (15)$$

C. Core Semantics

Below, we discuss quantitative semantics of the most widely-used Performance Tree operators. Semantics for the remaining operators can be found in Appendix A.

1) Identifying Sets of States using Atomic Proposition Composition: Many Performance Tree operators require sets of states as input; therefore, an elegant formalism-independent way for specifying these sets of states is necessary. This is done using atomic propositions that can be combined using boolean connectives, as demonstrated in Section IV. The allowed composition of atomic propositions is given by:

$$\Theta \overset{\text{def}}{=} \mathit{true} \mid \Theta \wedge \Theta \mid \neg\Theta \mid a$$

where $a \in \mathit{AP}$ is an atomic proposition label. Given a composition of atomic proposition labels A, we identify the corresponding set of states as $\{s \in \mathcal{S} : s \models A\}$ where the semantics of $s \models A$ is given by:

$s \models \mathit{true}$	for all s	$s \models \neg A$	iff $s \not\models A$
$s \models A$	iff $A \in \mathcal{L}(s)$	$s \models A \wedge B$	iff $s \models A \wedge s \models B$

2) PTD operator: The basic version of the passage time density operator takes two *States* nodes as input, which define the passage in terms of start and target states, and returns a passage time density function. Hence, its typing is

$$[\![\mathit{PTD}]\!] \quad :: \quad \Theta \times \Theta \rightarrow (\mathbb{R} \rightarrow \mathbb{R})$$

Evaluation of a *PTD* operator yields:

$$\mathit{PTD}(\mathit{States}(A, \mathit{start}), \mathit{States}(B, \mathit{target})) = f_{IJ}(t)$$

where $f_{IJ}(t)$ is the probability density function of P_{IJ}, the first passage time from a set of source states $I = \{s \in \mathcal{S} : s \models A\}$ to a set of target states $J = \{s \in \mathcal{S} : s \models B\}$, i.e. the first time the system enters a state in J, given that it has started in I and at least one transition has occurred. Here,

$$P_{IJ} = \inf\{u > 0 : Z(u) \in J, N(u) > 0, Z(0) \in I\}$$

$f_{IJ}(t)$ can be found by numerically inverting Laplace transform $L_{IJ}(s)$.

A variant of the passage time density operator is one that incorporates excluded states, in which case the typing becomes

$$[\![\mathit{PTD}]\!] \quad :: \quad \Theta \times \Theta \times \Theta \rightarrow (\mathbb{R} \rightarrow \mathbb{R})$$

Evaluation of a *PTD* operator with excluded states yields:

$$\mathit{PTD}(\mathit{States}(A, \mathit{start}), \mathit{States}(B, \mathit{target}), \mathit{States}(C, \mathit{excl})) \\ = f_{IJK}(t)$$

where $f_{IJK}(t)$ is the probability density function of $P_{IJK}(t)$, the first passage time between the set of states $I = \{s \in \mathcal{S} : s \models A\}$ and the set of states $J = \{s \in \mathcal{S} : s \models B\}$, constrained by the set of states $K = \{s \in \mathcal{S} : s \models C\}$, which must not form part of the passage. Here:

$$\begin{aligned} P_{IJK} &= \inf\{u > 0 : Z(u) \in J, N(u) > 0, \\ &\quad \forall u' < u. Z(u') \notin K, Z(0) \in I\} \end{aligned}$$

3) ProbInInterval operator: This operator represents the probability with which a passage takes place in a given amount of time. The operator has two inputs: a passage time density function, defining the passage, and a time range. It returns a probability. The typing for the operator is therefore

$$[\![\mathit{ProbInInterval}]\!] \quad :: \quad (\mathbb{R} \rightarrow \mathbb{R}) \times (\mathbb{R} \times \mathbb{R}) \rightarrow \mathbb{R}$$

Evaluation of a *ProbInInterval* operator yields:

$$\mathit{ProbInInterval}(\mathit{ptdf}, [\ldots](\mathit{Num}(r_1, \cdot), \mathit{Num}(r_2, \cdot))) = \\ \int_{r_1}^{r_2} \mathit{ptdf}(t)\, \mathrm{d}t$$

where ptdf refers to a passage-time density function.

4) ProbInStates operator: This operator represents the probability of being in a set of states at a particular time, having started from a given set of states. It requires three inputs: a composition of atomic propositions identifying the set of start states, a composition of atomic propositions identifying the set of target states and the time instant at which to consider the state of the model. Its typing is therefore

$$[\![\mathit{ProbInStates}]\!] \quad :: \quad \Theta \times \Theta \times \mathbb{R} \rightarrow \mathbb{R}$$

Evaluation of a *ProbInStates* operator yields:

$$\mathit{ProbInStates}(\mathit{States}(A, \mathit{start}), \mathit{States}(B, \mathit{target}), \\ \mathit{Num}(t, \mathit{time})) = \pi_{IJ}(t)$$

where $I = \{s \in \mathcal{S} : s \models A\}$ and $J = \{s \in \mathcal{S} : s \models B\}$, and the Laplace transform of $\pi_{IJ}(t)$ was defined in Section V-B3.

5) Moment operator: This operator represents a (raw) moment of a passage time density. It requires two inputs: an integer, representing which moment is to be calculated, and a passage time density that the moment is to be calculated from. The operator has type:

$$[\![\mathit{Moment}]\!] \quad :: \quad \mathbb{N} \times (\mathbb{R} \rightarrow \mathbb{R}) \rightarrow \mathbb{R}$$

Evaluation of a *Moment* operator proceeds via the Laplace transform as described in Section V-B1:

$$\mathit{Moment}(\mathit{Num}(n, \mathit{moment}), \mathit{ptdf}) = (-1)^n L^{(n)}(0)$$

where *ptdf* is a passage time density function and $L(s)$ is its Laplace transform.

6) SS:P operator: This operator calculates the steady-state probability distribution of an arbitrary state function over a set of states. That is, given a state function, $\mathcal{E}$, which associates a real value with every state in the system, the *SS:P* operator calculates the steady-state probability of $\mathcal{E}$ taking a particular value. *SS:P* requires two inputs: an atomic proposition expression, Θ, and a function, $\mathcal{E}$, on the set of states. $\mathcal{E}$ is represented by the *StateFunc* node in the performance tree notation. Θ puts a constraint on the set of states being considered. *SS:P* has the following type:

$$[\![\mathit{SS{:}P}]\!] \quad :: \quad \Theta \rightarrow \mathcal{E} \rightarrow (\mathbb{R} \rightarrow \mathbb{R})$$

where the $\mathcal{E}$ state function has the syntax:

$$\mathcal{E} \stackrel{\text{def}}{=} \#(a) \ \mid \ \mathcal{E} + \mathcal{E} \ \mid \ \mathcal{E} - \mathcal{E} \ \mid \ \mathcal{E} * \mathcal{E} \ \mid \ \mathcal{E}/\mathcal{E} \ \mid \ f(\mathcal{E}) \ \mid \ r$$

Here $a \in AP$ and $[\![f]\!] :: \mathbb{R} \to \mathbb{R}$ is a user-definable arbitrary real-valued function, and $r \in \mathbb{R}$. The semantics of the state function $\mathcal{E}$ are given by the evaluation function, $Eval(\mathcal{E}, s)$, which calculates the value of the function for a state, s:

$$\begin{array}{lcl}
Eval(\#(a), s) & = & \text{evaluation of } a \in AP \text{ in state } s \\
Eval(\mathcal{E}_1 + \mathcal{E}_2, s) & = & Eval(\mathcal{E}_1, s) + Eval(\mathcal{E}_2, s) \\
Eval(\mathcal{E}_1 - \mathcal{E}_2, s) & = & Eval(\mathcal{E}_1, s) - Eval(\mathcal{E}_2, s) \\
Eval(\mathcal{E}_1 * \mathcal{E}_2, s) & = & Eval(\mathcal{E}_1, s) \times Eval(\mathcal{E}_2, s) \\
Eval(\mathcal{E}_1 / \mathcal{E}_2, s) & = & Eval(\mathcal{E}_1, s) / Eval(\mathcal{E}_2, s) \\
Eval(f(\mathcal{E}), s) & = & f(Eval(\mathcal{E}, s)) \\
Eval(r, s) & = & r \qquad \text{:for all } s
\end{array}$$

Finally, *SS:P* can be defined as creating the probability mass function of the random variable Z_B, where Z_B represents the value of state function B on a state:

$$\textit{SS:P}(States(A), StateFunc(B)) = \mathbb{P}(Z_B = r)$$

where

$$\mathbb{P}(Z_B = r) = \begin{cases} \displaystyle\sum_{\substack{s\,:\,s\,\models A, \\ Eval(B,s)=r}} \pi_s & \text{:iff } r \in \{Eval(B, s) : s \models A\} \\ 0 & \text{:otherwise} \end{cases}$$

and π_s is the steady-state probability of state s.

7) FR operator: This operator represents the average rate of occurrence of any one of a group of actions (in the context of stochastic Petri nets, this is the average firing rate of a group of transitions). It requires a single input, namely the set of relevant actions. The typing of the operator is

$$[\![\textit{FR}]\!] \quad :: \quad 2^{\mathcal{A}} \to \mathbb{R}$$

Evaluation of a *FR* operator yields:

$$\textit{FR}(Actions(A)) = \sum_{a \in A} \sum_{X_i \text{:enables } a} R_a(X_i)\pi(X_i)$$

where $R_a(X_i)$ is the occurrence rate of action a in state X_i.

VI. Conclusion

In this paper, we have contrasted aspects of two formalisms for performance query specification, namely Performance Trees and CSL. We have discussed why Performance Trees may be more applicable to certain performance analysis scenarios – mainly due to its extended expressiveness – in the context of a model of a healthcare system.

With regards to future work, we aim to develop an integrated stochastic performance analysis toolset with a Performance Tree front-end and a Grid-based computational back-end that integrates several efficient numerical algorithms for passage time, transient and steady-state analysis. Once a Grid-enabled analysis back-end is in place, we will be able to model and analyse large systems of industrial scale.

Appendix

Table II gives semantics for remaining basic operators.

A. Dist operator

The passage time distribution operator transforms a passage time density into a passage time distribution.

$$[\![\textit{Dist}]\!] \quad :: \quad (\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R})$$

$$\textit{Dist(PTD(E))} = \mathbb{P}(0 < T_E \leq t) = \int_0^t f_E(t)\ dt$$

where T_E is the random variable corresponding to the first passage time of the passage defined by *E*, with a pdf of $f_E(t)$.

B. Conv operator

This operator represents the convolution of two passage time densities $\textit{ptdf}_1$ and $\textit{ptdf}_2$.

$$[\![\textit{Conv}]\!] \quad :: \quad (\mathbb{R} \to \mathbb{R}) \times (\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R})$$

$$\textit{Conv}(f(t), g(t)) = \int_0^t f(\tau)\, g(t - \tau)\, \mathrm{d}\tau$$

where $f(t) = \textit{ptdf}_1$ and $g(t) = \textit{ptdf}_2$ Section V-B1 for the Laplace construction of the convolution.

C. StatesAtTime operator

This operator selects the states that the system can occupy at a given time instant within a probability range.

$$[\![\textit{StatesAtTime}]\!] \quad :: \quad \mathbb{R} \times \mathbb{R} \times \mathbb{R} \to 2^{\mathcal{S}}$$

$$\textit{StatesAtTime(Num(t,time),[}\ldots\textit{](Num(}p_1\textit{,prob),Num(}p_2\textit{,prob)))} = \left\{ s : S \ \mid \ p_1 \leq \textit{ProbInStates(}\cdot\textit{,s,t)} \leq p_2 \right\}$$

D. SS:S operator

This operator selects the set of states that have a steady-state probability of a certain value, represented by an interval.

$$[\![\textit{SS:S}]\!] \quad :: \quad \Theta \times \mathbb{R} \times \mathbb{R} \to 2^{\mathcal{S}}$$

$$\textit{SS:S(States(I,start), [}\ldots\textit{](Num(}p_1\textit{,prob),Num(}p_2\textit{,prob)))} = \left\{ s : \mathcal{S} \ \mid \ p_1 \leq \textit{SS:P(s)} \leq p_2 \right\}$$

E. InInterval operator

This operator checks whether a given numerical value lies in a particular interval.

$$[\![\textit{InInterval}]\!] \quad :: \quad \mathbb{R} \times (\mathbb{R} \times \mathbb{R}) \to \mathcal{B}$$

$$\textit{InInterval(F,[}\ldots\textit{](Num(}r_1\textit{,}\cdot\textit{),Num(}r_2\textit{,}\cdot\textit{)))} = \begin{cases} true & \text{iff } r_1 \leq F \leq r_2 \\ false & \text{otherwise} \end{cases}$$

where $F = \textit{ProbInInterval(E)} \ \mid \ \textit{Moment(E)} \ \mid \textit{FR(E)} \ \mid \ +\textit{(E)} \ \mid \ -\textit{(E)} \ \mid \ *\textit{(E)} \ \mid \ /\textit{(E)} \ \mid \ \hat{}\textit{(E)} \ \mid \ \textit{SS:P(E)} \ \mid \ \textit{ProbInStates(E)}$.

Operation Node	Description
?	Represents the overall result of a performance query. It is the topmost node in every tree.
;	Multiple independent queries can be joined together by this node, so that only one query is submitted for analysis. It represents a vector of results of the independent queries.
PTD	Represents a passage time density, calculated from a given set of start and target states, as well as optional additional constraints on excluded states.
Dist	Represents a passage time distribution that is obtained from a passage time density.
Conv	Represents a convolution of two passage time densities.
ProbInInterval	Represents the probability with which a passage takes place in a certain amount of time.
ProbInStates	Represents the transient probability of the system being in a given set of states at a given instant in time.
Moment	Represents a specified raw moment of a passage time density.
FR	Represents the mean occurrence of an action / firing rate of a transition.
SS:P	Represents the steady-state probability distribution for a given set of states.
SS:S	Represents a set of states that have a certain steady-state probability.
StatesAtTime	Represents the set of states that the system can occupy at a given time.
InInterval	A boolean operator that determines whether a numerical value is within an interval or possibly within multiple intervals.
$\subseteq$	A boolean operator that determines whether a set is included in or corresponds to another set.
$\vee,\ \wedge$	Represent a boolean disjunction or conjunction of two logical expressions.
$\neg$	Represents boolean negation of a logical expression.
$>,\ \geq,\ ==,\ \leq,\ <$	Represent arithmetic comparisons of two numerical values.
$+,\ -,\ *,\ /,\ \hat{}$	Represent arithmetic operations on two numerical values.

Value Node	Description
States	Represents a set of states (identified by state labels).
Actions	Represents a set of actions (identified by action labels).
Num	Represents a real number.
Bool	Represents a boolean value.
$[\![\ldots]\!]$	Represents a numerical range.
StateFunc	Represents a function applied to a state that returns a real number.

Tab. I
DESCRIPTION OF PERFORMANCE TREE OPERATION AND VALUE NODES

Operation	Syntax	Semantics
$\subseteq$	$2^{AP} \times 2^{AP} \to \mathcal{B}$	$\subseteq(x,y) = \begin{array}{ll} \mathit{true} & \text{iff } x \subseteq y \\ \mathit{false} & \text{otherwise} \end{array}$
$>$	$\mathbb{R} \times \mathbb{R} \to \mathcal{B}$	$>(x,y) = \begin{array}{ll} \mathit{true} & \text{iff } x > y \\ \mathit{false} & \text{otherwise} \end{array}$
$\geq$	$\mathbb{R} \times \mathbb{R} \to \mathcal{B}$	$\geq(x,y) = \begin{array}{ll} \mathit{true} & \text{iff } x \geq y \\ \mathit{false} & \text{otherwise} \end{array}$
$<$	$\mathbb{R} \times \mathbb{R} \to \mathcal{B}$	$<(x,y) = \begin{array}{ll} \mathit{true} & \text{iff } x < y \\ \mathit{false} & \text{otherwise} \end{array}$
$\leq$	$\mathbb{R} \times \mathbb{R} \to \mathcal{B}$	$\leq(x,y) = \begin{array}{ll} \mathit{true} & \text{iff } x \leq y \\ \mathit{false} & \text{otherwise} \end{array}$
$==$	$\mathbb{R} \times \mathbb{R} \to \mathcal{B}$	$==(x,y) = \begin{array}{ll} \mathit{true} & \text{iff } x == y \\ \mathit{false} & \text{otherwise} \end{array}$
$\wedge$	$\mathcal{B} \times \mathcal{B} \to \mathcal{B}$	$\wedge(x,y) = \begin{array}{ll} \mathit{true} & \text{iff } x = \mathit{true} \text{ and } y = \mathit{true} \\ \mathit{false} & \text{otherwise} \end{array}$
$\vee$	$\mathcal{B} \times \mathcal{B} \to \mathcal{B}$	$\vee(x,y) = \begin{array}{ll} \mathit{true} & \text{iff } x = \mathit{true} \text{ or } y = \mathit{true} \\ \mathit{false} & \text{otherwise} \end{array}$
$\neg$	$\mathcal{B} \to \mathcal{B}$	$\neg(x) = \begin{array}{ll} \mathit{true} & \text{iff } x = \mathit{false} \\ \mathit{false} & \text{otherwise} \end{array}$
$+$	$\mathbb{R} \times \mathbb{R} \to \mathbb{R}$	$+(x,y) = x + y$
$-$	$\mathbb{R} \times \mathbb{R} \to \mathbb{R}$	$-(x,y) = x - y$
$*$	$\mathbb{R} \times \mathbb{R} \to \mathbb{R}$	$*(x,y) = x * y$
$/$	$\mathbb{R} \times \mathbb{R} \to \mathbb{R}$	$/(x,y) = x/y$
$\hat{}$	$\mathbb{R} \times \mathbb{R} \to \mathbb{R}$	$\hat{}(x,y) = x^y$
?	$E \to E$	$?(E) = E$
;	$E \times \ldots \times E \to [E, \ldots, E]$	$;(q_1, \ldots, q_n) = [r_1, \ldots, r_n]$ where $[r_1, \ldots, r_n]$ is a vector of the individual results of the queries $(q_1, \ldots, q_n)$ respectively.

Tab. II
SEMANTICS FOR REMAINING PERFORMANCE TREE OPERATORS

References

[1] R. Pyke, "Markov renewal processes: Definitions and preliminary properties," *Annals of Mathematical Statistics*, vol. 32, no. 4, pp. 1231–1242, December 1961.

[2] F. Bause and P. S. Kritzinger, *Stochastic Petri Nets - An Introduction to the Theory*. Vieweg Verlag, Wiesbaden, Germany, 1995.

[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, no. 2, pp. 93–122, May 1984.

[4] P. G. Harrison and N. M. Patel, *Performance Modelling of Communication Networks and Computer Architectures*, ser. International Computer Science Series. Addison Wesley, 1993.

[5] I. Mitrani, *Probabilistic Modelling*. Cambridge University Press, 1998.

[6] J. Hillston, "A compositional approach to performance modelling," Ph.D. dissertation, Department of Computer Science, University of Edinburgh, Edinburgh EH9 3JZ, UK, 1994, cST–107–94.

[7] ——, *A Compositional Approach to Performance Modelling*, ser. Distinguished Dissertations in Computer Science, 1996, vol. 12.

[8] R. Milner, *A Calculus of Communicating Systems*, ser. Lecture Notes in Computer Science. Springer-Verlag, 1980, vol. 92.

[9] ——, *Communication and Concurrency*, ser. PHI Series in Computer Science. Prentice Hall, 1989, iSBN 0 13 115007 3.

[10] H. Hermanns, "Interactive Markov chains," Ph.D. dissertation, Universität Erlangen-Nürnberg, July 1998.

[11] G. Ciardo, J. K. Muppala, and K. S. Trivedi, "SPNP: Stochastic Petri Net Package," in *PNPM'89, Proc. 3rd Intl. Workshop on Petri Nets and Performance Models*, 1989, pp. 142–151.

[12] G. Clark, T. Courtney, D. Daly, D. D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster, "The Möbius modeling tool," in *Proceedings the 9th International Workshop on Petri Nets and Performance Models*, B. Haverkort and R. German, Eds. IEEE Computer Society Press, Aachen, September 2001, pp. 241–250.

[13] W. J. Knottenbelt, "Generalised Markovian analysis of timed transitions systems," M.Sc. Thesis, University of Cape Town, South Africa, July 1996.

[14] N. J. Dingle, W. J. Knottenbelt, and P. G. Harrison, "HYDRA: HYpergraph-based Distributed Response-time Analyser," in *PDPTA'03, Proceedings of the 2003 International Conference on Parallel and Distributed Processing Techniques and Applications*, H. R. Arabnia and Y. Man, Eds., vol. 1, Las Vegas, NV, June 2003, pp. 215–219.

[15] C. Hirel, R. Sahner, X. Zhang, and K. S. Trivedi, "Reliability and performability modeling using SHARPE 2000," in *TOOLS 2000, Proc. 11th Intl. Conf. on Computer Performance Evaluation, Modelling Techniques and Tools*, ser. LNCS, vol. 1786, 2000, p. 345.

[16] P. Buchholz, J.-P. Katoen, P. Kemper, and C. Tepper, "Model-checking large structured Markov chains," *Journal of Logic and Algebraic Programming*, vol. 56, pp. 69–96, 2003.

[17] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with PRISM: A hybrid approach," in *TACAS'02, Proc. Tools and Algorithms for Construction and Analysis of Systems*, ser. LNCS, vol. 2280, 2002, pp. 52–66.

[18] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle, "A tool for model checking Markov chains," *Software Tools for Technology Transfer*, vol. 4, no. 2, pp. 153–172, 2002.

[19] J.-P. Katoen, M. Khattri, and I. S. Zapreev, "A Markov reward model checker," in *QEST'05, Proc. 2nd Intl. Conf. on the Quantitative Evaluation of Systems*, Italy, September 2005, pp. 243–244.

[20] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Verifying continuous-time Markov chains," in *Computer-Aided Verification*, ser. LNCS, vol. 1102, 1996, pp. 269–276.

[21] ——, "Model checking continuous-time Markov chains," *ACM Transactions on Computational Logic*, vol. 1, no. 1, pp. 162–170, 2000.

[22] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen, "On the logical characterisations of performability properties," in *Proc. ICALP 2000*, ser. LNCS, vol. 1853, 2000, pp. 780–792.

[23] ——, "Model-checking algorithms for continuous-time Markov chains," *IEEE Transactions on Software Engineering*, vol. 29, no. 6, pp. 524–541, June 2003.

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

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

[26] 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*, 2002, pp. 103–112.

[27] 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, Proc. Petri Nets and Performance Models*, University of Illinois at Urbana-Champaign, September 2003, pp. 62–71.

[28] T. Suto, J. T. Bradley, and W. J. Knottenbelt, "Performance trees: A new approach to quantitative performance specification," in *MASCOTS'06, Proc. 14^{th} Intl. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. Monterey, California, USA: IEEE Computer Society, September 2006.

[29] G. G. Infante López, H. Hermanns, and J.-P. Katoen, "Beyond memoryless distributions: Model checking semi-Markov chains," in *Proceedings of Process Algebra and Probabilistic Methods*, ser. Lecture Notes in Computer Science, L. de Alfaro and S. Gilmore, Eds., vol. 2165. Aachen: Springer-Verlag, September 2001, pp. 57–70.

[30] N. J. Dingle, P. G. Harrison, and W. J. Knottenbelt, "Uniformization and hypergraph partitioning for the distributed computation of response time densities in very large Markov models," *Journal of Parallel and Distributed Computing*, vol. 64, no. 8, pp. 908–920, August 2004.

[31] J. T. Bradley, N. J. Dingle, P. G. Harrison, and W. J. Knottenbelt, "Distributed computation of passage time quantiles and transient state distributions in large semi-Markov models," in *PMEO'03, Performance Modelling, Evaluation and Optimization of Parallel and Distributed Systems*. Nice: IEEE Computer Society Press, April 2003, p. 281.

[32] J. T. Bradley, N. J. Dingle, W. J. Knottenbelt, and H. J. Wilson, "Hypergraph-based parallel computation of passage time densities in large semi-Markov models," in *NSMC'03, Proceedings of the 4th International Workshop on Numerical Solutions of Markov Chains*, A. N. Langville and W. J. Stewart, Eds., University of Illinois at Urbana-Champaign, September 2003, pp. 99–120.

[33] J. T. Bradley, N. J. Dingle, S. T. Gilmore, and W. J. Knottenbelt, "Extracting passage times from PEPA models with the HYDRA tool: a case study," in *UKPEW'03, Proceedings of 19th Annual UK Performance Engineering Workshop*, S. A. Jarvis, Ed., University of Warwick, July 2003, pp. 79–90.

[34] J. T. Bradley and W. J. Knottenbelt, "The ipc/HYDRA tool chain for the analysis of pepa models," in *QEST'04, Proceedings of the 1st IEEE Conference on the Quantitative Evaluation of Systems*, B. Haverkort et al., Ed. University of Twente, Enschede: IEEE Computer Society Press, September 2004, pp. 334–335.

[35] S. W. M. Au-Yeung, N. J. Dingle, and W. J. Knottenbelt, "Efficient approximation of response time densities and quantiles in stochastic models," in *WOSP 2004, Proc. 4th International Workshop on Software and Performance*. Redwood City: ACM, January 2004, pp. 151–155.

[36] J. T. Bradley, N. J. Dingle, P. G. Harrison, and W. J. Knottenbelt, "Distributed computation of transient state distributions and passage time quantiles in large semi-Markov models," *Future Generation Computer Systems*, vol. 22, no. 7, pp. 828–837, August 2006.