# Feature Interaction in PEPA

Stephen Gilmore Jane Hillston[†]

$\{stg,\ jeh\}@dcs.ed.ac.uk$

**Abstract**

We consider the *feature interaction* problem in the context of the use of the PEPA stochastic process algebra. We introduce a notation for characterising a class of features and discuss its implementation.

## 1 Introduction

The *feature-oriented* approach to the specification and design of complex software systems offers great promise. The idea of a system which has been built by composing a number of well-engineered features appeals to both system designers—who are concerned with a *compositional* or structured approach to the construction of a system—and to system users—who wish to learn and understand complex systems in terms of substantive concepts. The widespread acceptance of the importance of features makes them a very desirable concept to build into a specification language since one of the uses of a specification language can be to provide a common working language between designers and users.

The *feature interaction problem* is concerned with the unexpected conflicts which can arise when a new feature is added to a system. Complex designs of all kinds can give rise to surprising interactions between features but the problem has been most intensively studied in the domain of telecommunications systems engineering. Modern network designs known as *intelligent networks* or *active networks* present significant feature interaction problems [1]. Feature interaction is the subject of an international workshop [2, 3].

What is a *feature*? The term does not have an accepted definition but we could characterise a feature as being a significant aspect or property of the system which could be used to compare this one to another. Most importantly, features are qualities which can *interact* and which can be *measured*. Features are not exclusively the province of telecommunications systems, both software and software-controlled systems have features which can interact. Taking the example of a lift system one would note that it has both *behavioural features* such as giving priority to requests from the executive floor and *performance features* such as average wait after pressing the call button. Our motivation for making this classification is so that we are able to consider the interaction of these two types of features.

The average waiting time for a lift is a measure of its quality of service. These types of measurements are a routine part of the assessment of the complex telecommunications networks which have fuelled the study of the feature interaction problem. Thus, to consider only behavioural features in this context with no regard for performance ones seems to us to only be increasing the likelihood of the sort of problems which happen whenever a behaviourally acceptable feature makes unreasonable demands on the system's performance. The problem of ensuring acceptable service in an increasingly sophisticated environment is a problem for present day telephony and, along with feature interaction, ensuring quality of service has been identified as a significant problem in the design of the next generation of networks [4].

The necessity of describing qualitative and quantitative information together leads us to judge unified notations for qualitative and quantitative modelling to be well suited to the analysis of the feature interaction problem. Examples of such unified notations include stochastic extensions of process algebras (such as PEPA [5]), stochastic extensions of Petri nets (such as GSPN [6]) or stochastic automata [7]. The feature interaction problem also provides a good setting for exemplary use of these formalisms. This arises because

[†]Laboratory for Foundations of Computer Science, The University of Edinburgh, Kings Buildings, Edinburgh EH9 3JZ. Telephone: +44 131 650 1000.

a significant difficulty in all quantitative modelling is the acquisition of reliable data about the duration of the significant activities of the system. This data must be accurate because it will be used in the calculation of performance measures of the system and the conclusions which are drawn from these calculations could be misleading if the data is inaccurate. However, in the context of the feature interaction problem we are considering the addition of a feature to an existing software system. In this setting we are able to monitor the system and acquire accurate performance data to be used in the calculations of performance measures before and after the new feature is added. In this way we can ensure the safe interplay of the new behavioural feature and the existing performance ones.

In the next section we give a very brief summary of the PEPA language and motivate the need for a formal notation for describing performance features of a PEPA model. We provide only a very brief summary of PEPA here: the reader should consult [5] for full details. In Section 3 we introduce a simple notation for describing performance features and in the following section we extend it to allow the description of more sophisticated measures. In Section 5 we show how the satisfaction of these performance measures can be established through the entension of the model with *testing components*. Conclusions and future directions for the work are at the end of the paper.

## 2 Relating features to a PEPA model

PEPA (Performance Evaluation Process Algebra) extends classical process algebra with the capacity to assign rates to the activities which are described in an abstract model of a system. It is a concise formal language with a small number of grammar rules which define the well-formed terms in the language. An activity of action type $\alpha$ performed at rate $r$ preceding $P$ is denoted by $(\alpha, r).P$. Using the symbol $\top$ instead of a rate denotes *passive* participation in a shared activity. Choices are separated by $+$. Cooperation between $P$ and $Q$ over $L$ is $P \underset{L}{\bowtie} Q$ or $P \parallel Q$ if $L$ is empty. Hiding the activities in $L$ is $P/L$. The notation for definitional equality is $\stackrel{\text{def}}{=}$. The syntax may be formally introduced by means of the following grammar:

$$
\begin{array}{rcl}
S & ::= & (\alpha, r).S \mid S + S \mid C_S \\
P & ::= & P \underset{L}{\bowtie} P \mid P/L \mid C
\end{array}
$$

where $S$ denotes a *sequential component* and $P$ denotes a *model component* which executes in parallel. $C$ stands for a constant which denotes either a sequential or a model component, as introduced in a definition. $C_S$ stands for constants which denote sequential components. The effect of this syntactic separation between these types of constants is to constrain legal PEPA components to be cooperations between sequential processes. PEPA is a high-level notation for Markov modelling because it is possible to generate directly from a PEPA model a continuous-time Markov process which faithfully encodes the temporal aspects of the PEPA model.

One reason to fix on a formal notation for a task such as performance modelling is to avoid misunderstanding and misinterpretation of a model. Of course, even when a notation is carefully defined, as PEPA is, there may still be errors of misrepresentation of parts of the system within the model but all of the users of the model can at least agree on the correct interpretation of a given model through recourse to the formal definition of the language. However, when we come to undertake a careful consideration of features of a model we see that we now have a need for a formal notation for *analysis* of performance models expressed in PEPA. Without this, we would not be able to state performance features precisely and such a shortfall would invalidate scientific consideration of the feature interaction problem.

The feature concept stresses the user-centred view of a system. A user makes an assessment of a system from the outside by collecting observations over time. Performance measures such as those which relate to perceived quality of service may be externally measured but there is another type of performance measure which must be observed by a component within the system itself. The difference is the following.

**Internally measurable features:** The measurement of these features requires the detection of activities which are not observable from a viewpoint outside the system. Such features would be of interest to the manager of the system. An example might be that one of the lifts should be utilised at least 50% of the time. The expression of the concept of utilisation would be likely to be based on the states of the system where the lift is moving. If the lift is programmed to home to a designated floor when it is not in use transporting passengers then it is likely that to capture this definition of utilisation the performance modeller would need to refer to operations other than the externally observable ones such as presses of call buttons.

**Externally observable features:** These may be measured by recording the observable activities which are performed by the system. These features are of interest to the users of the system. In the context of a lift a typical quality of service requirement might be that the average waiting time for a user on any floor will be less than three minutes. Another might be that 90% of calls from the executive floor will be satisfied in less than two minutes.

There is a further subtlety in the different examples of features which we have listed. It is our intention that each of these feature specifications should be expressed with reference to a PEPA model. This model is converted to an equivalent encoding as a Markov process for analysis. Different types of analysis of the Markov process are usually needed to calculate the different performance measures. The lift utilisation can be calculated from a *steady-state analysis* and the average waiting time can also be found in this way via an application of Little's Law but the percentage of calls satisfied in under two minutes requires a *transient analysis*. Steady-state analysis is done through the compilation of the infinitesimal generator matrix of the Markov process and the solution of this by Gaussian elimination or another technique from linear algebra. Transient analysis requires the modeller to use *uniformisation* or another technique.

We now consider specifying features of the steady-state probability distribution of the system using an internal view of the system.

# 3 A feature notation for equilibrium properties

We will begin by considering the description of performance features which can be computed by steady-state analysis. For this we will draw on a combination of standard mathematical notation together with the notation of the equivalence relations from stochastic process algebra. For example, we will specify the probability that component $P$ is in state $P_1$ thus: $\Pr(P = P_1)$. Such a specification is interpreted relative to a model of a system in which $P$ occurs and it succinctly describes the summation of the probabilities of the states of the system where sub-component $P$ is in state $P_1$. To be pedantic, we should write $\Pr(P \equiv P_1)$ if we intend to require $P$ to be literally $P_1$ and not just isomorphic to $P_1$. Similarly, we would write $\Pr(P \sim P_1)$ if we wished to denote the probability that $P$ is in a state which is bisimilar to $P_1$. These probabilities may then be used in further calculation such as $r \times \Pr(P = P_1)$ and those results used in comparisons as in $r \times \Pr(P = P_1) > M$. More complex descriptions of states may be expressed via logical operations as in $\Pr(P = P_1 \wedge Q = Q_1)$ or $\Pr(P = P_1 \vee P = P_2)$.

A process algebra allows the modeller to replicate components so that there may be, say, several copies of $P$ in the system description. Here, the effect of the feature specification must not be allowed to vary with the choice of the copy of $P$ which is considered because this would be a source of error for any modeller using the notation. Thus, we introduce a notion of *situation* (or *location*) of a copy of a component within a PEPA model. It could be the case that the component of interest occurs as a sub-component of another which has only one instance. For example, with

$$\begin{array}{rcl} R & \stackrel{\text{def}}{=} & P \underset{L}{\bowtie} M \\ S & \stackrel{\text{def}}{=} & P \underset{L}{\bowtie} Q \\ Sys & \stackrel{\text{def}}{=} & R \parallel S \end{array}$$

we refer to $R.P$ and $S.P$ to identify the two instances of $P$. If there is no convenient container for the component which we wish to designate then we shall simply resort to referring to the copy by number. We use the convention of numbering the copies of the components by following a pre-order traversal of the abstract syntax tree of the model, numbering copies of a component upwards beginning at 1. Thus we identify the left-hand copy of $P$ in $P \parallel P$ as $P\#1$ and the right-hand copy as $P\#2$.

This extended notation allows us to express feature specifications such as this: $\Pr(P\#1 \parallel P\#2 \sim P_1 \parallel P_2)$. This statement means that one of the copies of $P$ is in state $P_1$ and the other one is in state $P_2$, without specifying which is which.

$$
\begin{array}{lcll}
\varepsilon & ::= & \varepsilon + \varepsilon \mid \varepsilon - \varepsilon \mid \varepsilon \times \varepsilon \mid \varepsilon / \varepsilon & \text{(arithmetic expressions)} \\
& \mid & \varepsilon \geq \varepsilon \mid \varepsilon > \varepsilon \mid \varepsilon \leq \varepsilon \mid \varepsilon < \varepsilon & \text{(comparison expressions)} \\
& \mid & \mathbb{R} \mid \tau & \text{(constants and terms)} \\
\tau & ::= & \Pr(\phi) & \text{(probability terms)} \\
\phi & ::= & \phi \vee \phi \mid \phi \wedge \phi \mid \neg \phi & \text{(logical operators)} \\
& \mid & \sigma \equiv \sigma \mid \sigma = \sigma \mid \sigma \sim \sigma & \text{(local state conditions)} \\
& \mid & \alpha\uparrow \ \mid \alpha\uparrow\sigma & \text{(activity predicates)} \\
\sigma & ::= & \sigma \bowtie_{L} \sigma \mid \sigma / L \mid \sigma.C \mid C\#\mathbb{N} \mid C & \text{(situations)}
\end{array}
$$

Figure 1: Syntax of feature notation for equilibrium properties

One view of a stochastic process algebra model focuses on the states which it attains: another on the activities which it performs. In order to describe features in terms of the performance of activities we introduce a term for the probability that a type of activity is enabled. We use the notation $\Pr(\alpha\uparrow)$ for this, whenever the action type of the activity is $\alpha$. Thus the interpretation of an activity name as a predicate is that the predicate is satisfied whenever the model is in a state $S$ and there is both a state $S'$ and a rate $r$ such that $S \xrightarrow{(\alpha,r)} S'$. A convenient extension to this notation is $\Pr(\alpha \uparrow P)$, meaning that activity $\alpha$ could be performed by component $P$ of the model. However, we shall regard these two forms of *activity probability* as simply convenient abbreviations for a much more complex predicate where the components are constrained (or a given component is constrained) to those states where they may perform an activity of action type $\alpha$. The cases where the meanings of these two expressions would differ arise whenever the model is not *PEPA live* or not *fully live* so we restrict our consideration to fully live models from this point onwards. We have previously presented the definitions of these two notions of liveness in our adaptation to the stochastic process algebra setting of the structural theory of Petri nets [8]. We recap them here for completeness.

**PEPA liveness:** For each reachable state and every action type there exists another state which is reachable where an activity of that type can be performed.

**Full liveness** For each reachable state and each syntactic occurrence of an activity within a sequential component there exists another state which is reachable where this occurrence of the activity can be performed.

We now have performance feature *expressions* $\varepsilon$, probability *terms* $\tau$, *predicates* $\phi$ and *situations* $\sigma$. These are expressed in the syntax presented in Figure 1. The plus symbol is used in the syntax to denote real number addition and not the choice operator of process algebra. The division symbol is used to denote both real number division (in expressions) and the PEPA hiding operator (in situations). We write $\mathbb{R}$ to denote the real numbers and $\mathbb{N}$ to denote the natural numbers.

The characteristics of this feature notation are that it allows the modeller to inspect internal local states of model components and to consider the equilibrium probability of attaining significant states of interest. Under the interpretation of activity probabilities as abbreviations for more complex predicates over states we may consider this notation to be entirely based on model states. In the notation for transient properties we consider the *external* observation of activities.

# 4 A feature notation for transient properties

When we come to consider transient measures we need to modify our language of probabilistic expressions. In contrast to the steady state probabilities of being a state in which certain local state conditions are satisfied, e.g. $\Pr(P = P_1 \wedge Q = Q_1)$ or in which a particular action type is enabled $\Pr(\alpha\uparrow)$, we now wish to express conditions such as *the probability, starting from a state after $\alpha$ has been performed, that we reach*

$$
\begin{array}{rcll}
\varepsilon & ::= & \dots \textit{ as before } \dots & \text{(feature expressions)} \\
\tau & ::= & \Pr(n, t, \phi \mid \iota) & \text{(iteration probability)} \\
 & \mid & \Pr(n, t, \phi) & \text{(free iteration probability)} \\
 & \mid & \Pr(\phi \mathbin{@} t \mid \iota) & \text{(arrival probability)} \\
 & \mid & \Pr(\phi \mathbin{@} t) & \text{(free arrival probability)} \\
 & \mid & \mathrm{Ti}(\phi \mid \iota) & \text{(expected time)} \\
 & \mid & \Pr(\infty)(\phi) & \text{(long run probability)} \\
\iota = \phi & ::= & \phi \vee \phi \mid \phi \wedge \phi \mid \neg\phi & \text{(logical operators)} \\
 & \mid & \sigma \equiv \sigma \mid \sigma = \sigma \mid \sigma \sim \sigma & \text{(local state conditions)} \\
 & \mid & \alpha\!\downarrow & \text{(activity predicate)} \\
\sigma & ::= & \sigma \parallel \sigma \mid \mathit{Pass} \mid \mathit{Fail} \mid C_T & \text{(testing situations)}
\end{array}
$$

Figure 2: Syntax of feature notation for transient properties

*a state where $\beta$ has been performed, in less than $t$ time units.* For this purpose we introduce the following additional forms of probabilistic expressions, and clarify the notation for steady state expressions:

**Iteration probability:** We use the term $\Pr(n, t, \phi \mid \iota)$ to denote the probability, given that condition $\iota$ holds initially (meaning, at time $t = 0$), that the process can reach a state in which condition $\phi$ holds for the $n$th time in time less than or equal to $t$. In general we will be interested in the first time that the condition $\phi$ is met so by default $n = 1$ and in this case we omit the first clause from the expression. The conditions $\phi$ and $\iota$ are predicates over transient properties of the processes, and the initial description predicate $\iota$ can be omitted if we start from a state which is chosen at random.

**Arrival probability:** We use $\Pr(\phi \mathbin{@} t \mid \iota)$ to express the probability, given that condition $\iota$ holds initially, that after elapsed time $t$ the process is in a state which satisfies $\phi$. As above, the conditions $\iota$ and $\phi$ are predicates over transient properties, and once again the initial description predicate $\iota$ can be omitted if we start from a state which is chosen at random.

**Expected time:** The notation $\mathrm{Ti}(\phi \mid \iota)$ represents the expected time to reach a state in which the condition $\phi$ is satisfied given that the process starts in a state which satisfies $\iota$. The uses of the conditions $\phi$ and $\iota$ are the same as in the previous cases.

**Long run:** The long run or steady state probability that the process is in a state in which condition $\phi$ holds is written $\Pr(\infty)(\phi)$. This is the form of expression we considered in Section 3 and when it is clear from context we will omit the term $(\infty)$.

Summarising the above we have the feature notation for transient properties which is shown in Figure 2. The notation for predicates is now altered because we express only observations about the original system. We denote the second modality on activities ('has been performed') with the notation $\Pr(\alpha\!\downarrow)$. The notion of situation is modified to allow consideration of the states of distinguished *testing components* only. We use $C_T$ to represent a constant which identifies a testing component.

# 5 Testing satisfaction of feature specifications

Thus far it has been possible to present our exposition in the terms of the PEPA language and to refer to process algebra concepts such as states, local states or activities. When we wish to check that a particular model satisfies a feature specification it is necessary to first convert the model into its equivalent Markov process encoding. Transient analysis of an Markov process is not based on the straightforward solution of the global balance equations which are used to generate the steady state probability distribution. Instead, depending on the measure which must be calculated, modifications of the Markov process or its matrix representation may be necessary and other solution algorithms are needed. If we are only interested in transient properties of our process then we do not need the ergodicity condition which is necessary for a Markov process to attain steady-state equilibrium.

The objective of our work is to use specifications in the simple probabilistic expression notation outlined above to automatically generate process algebra components which perform stochastic tests. These testing components can then be appropriately composed with the original model and from this extended model the Markov process and the probability matrix which are needed for transient analysis can be generated using the algorithms which are already available for generation of the matrix which is used in steady-state analysis.

The algorithms for generating the Markov process representation of a PEPA model are already implemented in the PEPA Workbench [9] and of course it would also be possible to extend the Workbench implementation to include an alternative set of algorithms for the generation of the modified Markov processes which would be needed for transient analysis. As an alternative, it would be possible to extend the input to the Workbench to include switches which indicate whether steady-state or transient solution is intended and to provide the additional information which may be needed for transient analysis but our approach of generating additional testing components and using the existing facilities seems more elegant. A sufficiently expressive fragment of the feature notation for equilibrium properties has already been implemented in the PEPA State Finder (as described in [10]) so we now consider the implementation of the notation for transient properties.

### 5.1 Uniformisation

We base our transient analysis on the technique known as *uniformisation* or *randomisation*. This is briefly outlined below. Our presentation is based on [11].

Consider a continuous time Markov process $\{X(t), t \geq 0\}$ with infinitesimal generator matrix $\mathbf{Q}$ and starting state distribution vector $\pi(0)$. We assume that the exit rate from all states is bound by some value $R$, i.e. $|q_{ii}| \leq R$ for all states $i$ in the state space $S$. We choose $R = \max_i |q_{ii}|$. In transient analysis our objective is to find $\pi(t)$, the state probability distribution vector at time $t$. From this distribution various other expectations and measures can be calculated, just as we calculate steady-state measures based on the steady-state probability distribution $\pi$.

If the exit rate from every state is the same i.e. $|q_{ii}| = R$ for all $i$, then calculating $\pi(t)$ is relatively straightforward. The approach of uniformisation is to make all the $q_{ii}$ equal by introducing null-events into states as needed. These null-events have no effect, in that they do not make the process change state, but they do ensure that the exit rate from all states becomes equal. Thus for each state $i$, a *do-nothing* event is introduced with rate $R + q_{ii}$ (recall that $q_{ii}$ is negative).

Now the rate at which events occur within the system is independent of the state of the system and is $R$. The number of events in the interval from 0 to $t$ has a Poisson distribution with parameter $Rt$:

$$\Pr(k \text{ events in interval } (0,t)) = e^{-Rt}\frac{(Rt)^k}{k!}$$

Let $X_k$ be the state of the system after the $k$th event and let $X_0$ be the starting state: the process $\{X_k, k \geq 0\}$ is a discrete time Markov process over the same state space as $X(t)$. Moreover the transition probabilities $p_{ij}$ of $X_k$ are readily derived from the transition rates of $X(t)$:

$$\begin{aligned} p_{ij} &= \Pr(X_k = j \mid X_{k-1} = i) = q_{ij}/R \qquad i \neq j \\ p_{ii} &= 1 + \frac{q_{ii}}{R} \end{aligned}$$

We denote the probability of being in state $j$ after $k$ events by $\pi_j^k$ and note that $\pi_0^k = \pi_i(0)$. The $\pi_j^k$ can be calculated by standard techniques and

$$\begin{aligned} \pi_j(t) &= \sum_{k=0}^{\infty} \pi_j^k \Pr(k \text{ events in interval } (0,t)) \\ &= \sum_{k=0}^{\infty} \pi_j^k e^{-Rt}\frac{(Rt)^k}{k!} \end{aligned}$$

Clearly when carrying out a numerical evaluation the infinite sum must be truncated at some value $K$: from this we can bound the error introduced, $\epsilon_K$ [12]:

$$\epsilon_K \leq 1 - \sum_{k=0}^{K} e^{-Rt} \frac{(Rt)^k}{k!}$$

Conversely, for a chosen $\epsilon$ we can choose $K$ sufficiently large that

$$1 - \sum_{k=0}^{K} e^{-Rt} \frac{(Rt)^k}{k!} \leq \epsilon.$$

### 5.2 Iteration testing

In this subsection we consider the forms of stochastic test which must be added to the basic PEPA process in order to automatically generate the Markov process suitable for testing probabilistic expressions of the form $\Pr(n, t, \phi \mid \iota)$. In [13] Melamed and Yadin describe a technique for finding sojourn times based on transient solution via uniformisation. Essentially their approach relies on complementing the state space with a distinguished goal state, $\gamma$, which is entered when the sojourn is completed. Assuming that the sojourn began at time 0 then $\pi_\gamma(t)$ is the probability that the sojourn is shorter than $t$. We adopt a similar approach here. The distinguished state is introduced using an additional PEPA testing component which is composed with the original model. The idea of generating testing components to perform model checking by means of reachability analysis is of course not novel. An early reference is [14] and a more recent one is [15].

The intuition behind the design of the testing component is that it witnesses the starting condition $\iota$ and waits for the final condition $\phi$. When it has witnessed $\phi$ it enters a distinguished state which we call *Pass*. How we construct the testing component depends on the form of the conditions $\iota$ and $\phi$.

It may be the case that the starting condition $\iota$ is satisfied by more than one state of the process. If this is the case we need to find the distribution over these states and we take this to be our starting vector $\pi(0)$. To find this we calculate the steady-state distribution of the process and then identify those states in which $\iota$ holds, the subset $S(\iota)$. We then form the conditional distribution on the assumption that we are within this subset of states.

$$\pi_i(0) = \begin{cases} \dfrac{\pi_i}{\sum_{j \in S(\iota)} \pi_j} & \text{if } i \in S(\iota) \\ 0 & \text{if } i \notin S(\iota) \end{cases}$$

Using transient analysis based on uniformisation we then allow the model to evolve for a time equal to $t$ and then look at the probability that condition $\phi$ has been satisfied, i.e.

$$\sum_{i:\, Test = Pass} \pi_i(t).$$

Assume that our existing PEPA model is $M$ and that it is fully live. Our approach will be to reduce feature specifications of the form $\Pr(n, t, \phi \mid \iota)$ to one of the form $\Pr(\phi' \mathbin{@} t \mid \iota)$ where $\phi'$ is a derived condition in terms of the model and the test component which we have generated which, in general, will assert that the test has been passed. The model extended with the test component typically takes the form $M \underset{L}{\bowtie} \mathit{Test}$ and $\phi'$ will typically be $\mathit{Test} = \mathit{Pass}$ for positive predicates and $\mathit{Test} = \mathit{Fail}$ for negative ones.

### 5.3 Generating testing components

Figure 3 shows the generation of the testing components which are needed to encode predicates over the transient behaviour of the model. Initialisation predicates $\iota$ which mark the start of the test are incorporated into the conditional probability distribution as explained in Section 5.2. Thus the cases of free and non-free iteration can be treated symmetrically. Testing components are generated which passively witness the completion of activities thus: $(\alpha, \top).\mathit{Pass}_\alpha$. Simple tests such as these can be composed in order to build

$$
\begin{array}{llcl}
\textbf{(iteration)} & M \models \Pr(n, t, \alpha{\downarrow} \mid \iota) > p & \rightsquigarrow & M \underset{\{\alpha\}}{\bowtie} T \models \Pr((T = Pass_\alpha) \mathbin{@} t) > p \\
& \text{where } T & \stackrel{\text{def}}{=} & (\alpha, \top)^n.Pass_\alpha \\
& \text{and } Pass_\alpha & \stackrel{\text{def}}{=} & (\alpha, \top).Pass_\alpha \\
\\
\textbf{(negation)} & M \models \Pr(\neg\phi) > p & \rightsquigarrow & M \models \Pr(\phi) \leq 1 - p \\
\\
\textbf{(conjunction)} & M \models \Pr(\phi \wedge \psi) > p & \rightsquigarrow & M \underset{K \cup L}{\bowtie} (T_\phi \parallel T_\psi) \models \Pr(\phi' \wedge \psi') > p \\
& \text{if } K \cap L = \emptyset & & \\
& \text{and } M \models \Pr(\phi) > p & \rightsquigarrow & M \underset{K}{\bowtie} T_\phi \models \Pr(\phi') > p \\
& \text{and } M \models \Pr(\psi) > p & \rightsquigarrow & M \underset{L}{\bowtie} T_\psi \models \Pr(\psi') > p \\
\\
\textbf{(disjunction)} & M \models \Pr(\phi \vee \psi) > p & \rightsquigarrow & M \underset{K \cup L}{\bowtie} (T_\phi \parallel T_\psi) \models \Pr(\phi' \vee \psi') > p \\
& \text{if } K \cap L = \emptyset & & \\
& \text{and } M \models \Pr(\phi) > p & \rightsquigarrow & M \underset{L}{\bowtie} T_\phi \models \Pr(\phi') > p \\
& \text{and } M \models \Pr(\psi) > p & \rightsquigarrow & M \underset{K}{\bowtie} T_\psi \models \Pr(\psi') > p
\end{array}
$$

Figure 3: Generating testing components for a model

up the more complex predicates which count the number of occurrences of the activity of interest. These construct a repeated sequential prefix of the required number of observations of the designated activity, say

$$\underbrace{(\alpha, \top).(\alpha, \top) \ldots (\alpha, \top)}_{n \text{ times}}.Pass_\alpha$$

We introduce the abbreviation $(\alpha, \top)^n.Pass_\alpha$ to stand for the above expression.

We decompose compound predicates and recompose those which we generate in order to recursively unfold conjunction and disjunction and implement negation via subtraction. Thus, for example, to re-express the predicate $\alpha{\downarrow} \wedge \beta{\downarrow}$ we generate a testing component for $\alpha{\downarrow}$, $T_{\alpha\downarrow}$, and a testing component for $\beta{\downarrow}$, $T_{\beta\downarrow}$, and our new predicate would be

$$T_{\alpha\downarrow} = Pass_\alpha \wedge T_{\beta\downarrow} = Pass_\beta.$$

Treating compound predicates by decomposing them into a flat collection of testing components could lead to confusion in the assessment of compound properties where an activity name is used more than once. For example, given the predicate $(\alpha{\downarrow} \wedge \beta{\downarrow}) \vee (\beta{\downarrow} \wedge \gamma{\downarrow})$ performing $\alpha$ and $\beta$ does not guarantee that the generated predicate will be satisfied because these activities might lead to the testing component reaching the state

$$((Pass_\alpha \parallel (\beta, \top).Pass_\beta) \parallel (Pass_\beta \parallel (\gamma, \top).Pass_\gamma))$$

instead of

$$((Pass_\alpha \parallel Pass_\beta) \parallel ((\beta, \top).Pass_\beta \parallel (\gamma, \top).Pass_\gamma)).$$

We impose the strict condition that predicates may not *interfere* in this way by requiring that their sets of free activity names are disjoint.

## 6 Further work

We have focused here on performance features. The challenge when adding new behavioural features to an existing model is to hygienically separate the specification of the new feature from the existing system and to show how these can be integrated into the given structure. The study of the *feature integration* problem in the stochastic process algebra context remains as further work. Assessing the utility of the present notation for specifying features by conducting a case study in modelling a practical software system also remains as further work.

The notations for feature specification presented here could be integrated with one based on the logical description of specification of rewards. Such a notation has been created previously for PEPA via an extended application of the modal $\mu$-calculus [16]. The symmetric integration of the two notations and the consideration of their use together remains as future work.

## 7 Conclusions

We have presented a notation for the description of feature specifications which relate to stochastic process algebra models expressed in the PEPA modelling notation. Our notation for feature specification separates into two parts, the first for the specification of internally measurable quantities of the system's equilibrium state, and the second for externally observable patterns in the system's transient behaviour. We have shown a method of reducing specifications in the latter to specifications in the former by extending the system model with testing components.

There are many design decisions to be taken in the creation of any such specification notation. For the specification of transient properties we restricted ourselves to observing only the activities which are performed by the system whose features we are studying. This discipline means that our feature specifications have the desirable property that they are not dependent on the internal structure of the model and could be used across a range of models to compare them in terms of the chosen feature. It would have been unwise to similarly restrict ourselves to observing only external activities of the testing components themselves since these activities must have rates and they must of course be performed at a much faster rate than the other activities of the model in order not to compromise the accuracy of the test itself. Such a wide disparity between rates of performing activities introduces the well-known *stiffness* problem for Markov processes and would have been a barrier to their efficient solution although the isolation of the faster activities in the testing components would provide a promising setting for the use of time-scale decomposition [17] in the solution of these extended models. Thus this design decision of the feature specification notation seems to be justified.

### Acknowledgements

Stephen Gilmore is supported by the 'Distributed Commit Protocols' grant from the EPSRC and by Esprit Working group FIREworks. Jane Hillston is supported by the ESPRC 'COMPA' grant. The authors are grateful to Nigel Thomas and the anonymous referees for PAPM for helpful comments on an earlier version of the paper.

## References

[1] C. Capellmann, R. Demant, R. Galvez-Estrada, U. Nitsche, and P. Ochsenschläger. Verification by behaviour abstraction: A case study of service interaction detection in intelligent telephone networks. In *Computer Aided Verification*, Springer, LNCS 1102, pages 466–469, 1996.

[2] K. E. Cheng and T. Ohta, editors. *Feature Interactions in Telecommunications III*, Tokyo, Japan, October 1995. IOS Press.

[3] P. Dini, R. Boutaba, and L. Logrippo, editors. *Feature Interactions in Telecommunications Networks IV*, Montreal, Canada, 1997. IOS Press.

[4] S. Bhattacharjee, K. Calvert, and E.W. Zegura. On active networking and congestion. Technical Report GIT-CC-96-02, College of Computing, Georgia Tech., 1996.

[5] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[6] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley, 1995.

[7] P.R. D'Argenio, J-P. Katoen, and E. Brinksma. A stochastic automata model and its algebraic approach. In E. Brinksma and A. Nymeyer, editors, *Proceedings of the Fifth Annual Workshop on Process Algebra and Performance Modelling*, pages 1–16. University of Twente, June 1997. Centre for Telematics and Information Technology, Technical report number 97-14.

[8] S. Gilmore, J. Hillston, and L. Recalde. Elementary structural analysis for PEPA. Technical Report ECS-LFCS-97-377, Laboratory for Foundations of Computer Science, Department of Computer Science, The University of Edinburgh, 1997.

[9] S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In *Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, number 794 in Lecture Notes in Computer Science, pages 353–368, Vienna, May 1994. Springer-Verlag.

[10] G. Clark, S. Gilmore, J. Hillston, and N. Thomas. Experiences with the PEPA performance modelling tools. In R.J. Pooley and N. Thomas, editors, *Proceedings of the Fourteenth UK Performance Engineering Workshop*, Edinburgh, Scotland, July 1998. To appear in a special issue of *IEE Software*, December 1998.

[11] W. Grassmann. Finding transient solutions in Markovian event systems through randomization. In W.J. Stewart, editor, *Numerical Solution of Markov Chains*, volume 8 of *Probability: Pure and Applied*, chapter 18, pages 357–371. Marcel Dekker, 1991.

[12] W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[13] B. Melamed and M.Yadin. Numerical computation of sojourn-time distributions in queueing networks. *ACM Transactions on Computer Systems*, 3(1):839–854, 1984.

[14] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. of the 1st. Annual Symposium on Logic in Computer Science, LICS'86*, pages 322–331. IEEE Computer Society Press, 1986.

[15] L. Aceto, A. Burgueño, and K. G. Larsen. Model checking via reachability testing for timed automata. In B. Steffen, editor, *Proceedings of TACAS '98,* Lisbon, volume 1384 of *Lecture Notes in Computer Science*, pages 263–280. Springer-Verlag, 1998.

[16] G. Clark. Formalising the specification of rewards with PEPA. In M. Ribaudo, editor, *Proceedings of the Fourth Annual Workshop on Process Algebra and Performance Modelling*, pages 139–160. Dipartimento di Informatica, Universitá di Torino, CLUT, July 1996.

[17] J. Hillston and V. Mertsiotakis. A simple time scale decomposition technique for stochastic process algebras. In S. Gilmore and J. Hillston, editors, *Proceedings of the Third International Workshop on Process Algebras and Performance Modelling*, pages 566–577. Special Issue of *The Computer Journal*, 38(7), December 1995.