Towards Automatic Derivation of Performance Measures from PEPA Models

Graham Clark* Jane Hillston*

Abstract

Stochastic process algebras, such as PEPA, provide a novel approach to performance modelling. As well as facilitating a compositional approach, process algebra models focus on a system's behaviour rather than its state space. Classical process algebras are complemented by *modal* and *temporal* logics which concisely express possible model behaviours. These logics are widely used during functional analysis to aid in the verification of system behaviour. During performance analysis we seek to *evaluate* rather than simply verify the behaviour of a system, and for performance models based on continuous time Markov processes, reward structures are commonly used for this purpose.

In this paper we describe a combination of these techniques—the PEPA reward language and its use to derive performance measures from PEPA models. The reward language is based on a modal logic which characterises specific behaviours within the PEPA model and may be used to develop a reward structure over the underlying Markov process. A prototype implementation exists within the PEPA Workbench.

1 Introduction

Over the last decade there has been growing acceptance of the verification of a system's functional behaviour during design, using formal techniques such as those involving process algebra models and their complementary modal logics. Unfortunately verifying the system's temporal behaviour, or performance, is still often neglected until implementation is well under way.

*Department of Computer Science, The University of Edinburgh, Kings Buildings, Edinburgh, EH9 3JZ. Email: {gcla, jeh}@dcs.ed.ac.uk

This contrast has been one of the motivations behind the recent development of stochastic process algebras (SPA), which integrate performance analysis with functional analysis [Her90, GHR93, Hil94]. These languages are process algebras which are enhanced with information about duration of activities and, via a race policy, their relative probabilities.

As mentioned above, classical process algebras are complemented by modal logics, which formally express properties of a system in terms of its behaviour. Verifying that a system possesses a particular logical property is called *model checking* [SW89]. The development of efficient algorithms to test the qualitative behaviour of a model is still an active area of research.

Although one of the strengths of SPA languages is their formality and the support that this provides for automated reasoning, deriving performance measures from SPA models is currently carried out in an ad hoc manner. At best, an informal approach, based on imposing a *reward structure* over the underlying Markov process, is used. In this paper we utilise a simple modal logic for one SPA, Hillston's PEPA. This is used in a technique which has been developed with the explicit intention of supporting behavioural reasoning about the quantitative behaviour of systems. At the level of the underlying Markov process, rewards are still used to calculate performance measures. The novel aspect of the work is that the specification of measures to be calculated takes place at the level of the stochastic process algebra, the high level modelling paradigm. Moreover it is given in terms of the *behaviour* of the system, in keeping with the process algebra approach. The result is the PEPA reward language, which uses the logic to define a reward structure, and is described in detail in Section 4. A prototype implementation has been developed to work in conjunction with the PEPA Workbench [GH94].

A similar approach, using a *temporal logic*, was recently proposed by Hermanns. In [Her] he proposes the use of logic formulae to partition a process algebra model, each partition exhibiting a particular behaviour. Thus, whereas a temporal logic formula is used to verify the functional behaviour of a classical process algebra model, Hermanns considered using a formula to discriminate between stages of a process's life in which it could, and could not, exhibit a particular behaviour.

The rest of the paper is organised as follows. In Section 2, PEPA, and its use for performance modelling, are briefly revised. The current approach to deriving performance measures from PEPA models, and its problems, are described in Section 3. The main contribution of this paper is the use of the logic in the language presented in Section 4. We demonstrate the use of this language in the examples presented in Section 5 and conclude the paper with an outline of future work in Section 6.

2 Performance Modelling using PEPA

Classical process algebras such as CCS [Mil89] and CSP [Hoa85] disregard the notion of time, and model functional behaviour only. PEPA extends these algebras by associating a random variable, which represents a duration, with every action type. Therefore activities take time, and the performance of components can be studied. The random variables are chosen to be exponentially distributed to give a clear relationship between the SPA model and a continuous time Markov process. It is from the steady state solution of the underlying Markov process that performance measures can be calculated.

PEPA models are constructed from components, which are able to interact with each other. Each of these components is capable of performing activities. Formally, an activity $a \in \mathcal{A}ct$ is described as a pair (α, r), where $\alpha \in \mathcal{A}$ is the type of the activity, known as the *action type*, and $r \in \mathbb{R}^+$ is the activity rate. $\mathbb{R}^+$ is defined as the set of positive real numbers together with the symbol $\top$. This symbol denotes an undefined activity rate. Activities may only take place in synchrony with another activity of the same type. In a complete model all such synchronisations must include at least one activity whose rate is not undefined, otherwise the model cannot be solved. When r is defined, its value is the parameter of an exponential distribution and thus governs the duration of the activity. The set of all syntactic action types will be denoted by $\mathsf{action\text{-}type}$.

In typical process algebra style, a small set of combinators is used to construct larger process algebra terms from smaller ones—this is the compositional approach. These combinators include sequential composition $((\alpha, r).P)$, synchronisation $(P \underset{L}{\bowtie} Q)$, encapsulation (P/L), and equational definition $(P \stackrel{\mathrm{def}}{=} Q)$. The set of all syntactically well-formed PEPA terms will be denoted by $\mathsf{process}$.

Example The following simple example illustrates the use of the combinators and the compositional style of modelling.

A worker process requires the use of a resource, idles and then evolves back into a worker process, i.e. it cycles. In the PEPA representation we combine each of its action types (`holding` and `idling`) with the corresponding duration distributions, from the perspective of the worker. The recursive nature of the defining equation reflects the cyclic pattern of behaviour described above.

$$Worker \stackrel{\mathrm{def}}{=} (\mathtt{holding}, whr).(\mathtt{idling}, ir).Worker$$

The resource has a similar pattern of behaviour—it is either being used by a worker, or it is engaging in an updating activity in preparation for being used again. The defining equation for the resource, shown below, is very similar to that for the worker.

$$Resource \stackrel{\mathrm{def}}{=} (\mathtt{holding}, rhr).(\mathtt{update}, ur).Resource$$

Note, however, that the resource's representation of the holding activity reflects the duration of the action from its own perspective which may differ from that of the worker. The components $Worker$ and $Resource$ are the basic components of our system. The complete model is formed by specifying how these components interact with each other.

$$System \stackrel{\mathrm{def}}{=} Worker \underset{\{holding\}}{\bowtie} Resource$$

The annotation of the synchronisation combinator $\bowtie$, in this case $\{\mathtt{holding}\}$, indicates that the components must synchronise on the specified activities. They may proceed concurrently on all other activities.

Representing the system as a combination of separate components means that we can easily extend our model. For example if there were two workers competing for use of the resource, the new system would be represented as follows:

$$System' \stackrel{\mathrm{def}}{=} (Worker \parallel Worker) \underset{\{holding\}}{\bowtie} Resource$$

where $\parallel$ is used to denote the pure parallel combinator, $\underset{\emptyset}{\bowtie}$. Alternatively if we wished to embed the original system into a more complex environment, but ensure that the interaction between the worker and the resource could not be affected by the environment, we would make use of the encapsulation combinator to hide the holding activity:

$$Large\_System \stackrel{\mathrm{def}}{=} \Big(System/\{\mathtt{holding}\}\Big) \underset{L}{\bowtie} Environment$$

Hiding an activity means that its action type appears to the environment as the internal delay, τ. Activities of such a type cannot be used by a process to synchronise. Therefore the behaviour of $System/\{\mathtt{holding}\}$ in $Large\_System$ will be the same regardless of whether or not $\mathtt{holding} \in L$.

2.1 Derivatives and the Underlying Markov Process

If a PEPA process P may perform an (α, r) activity and evolve into Q, we write $P \xrightarrow{(\alpha,r)} Q$, and say Q is a (one-step) *derivative* of P. The relation denoted by $\xrightarrow{(\alpha,r)}$ is called the *transition relation.* Further, for any PEPA process P, the *derivative set* of P, $ds(P)$, is the least set of derivatives closed under the $\xrightarrow{(\alpha,r)}$ relation, and it thus captures all the reachable states of the system.

PEPA has a formal operational semantics and this associates a labelled *multi*-transition system with each PEPA process expression [Hil94]. A labelled transition system is a triple $(S, T, \longrightarrow)$ where S is a set of states, T a set of transition labels, and $\longrightarrow \subseteq S \times T \times S$ is the transition relation. With PEPA, the states are syntactic process expressions, the transition labels are the (α, r) activities, and the transition relation is given by the operational rules. However a multi-transition relation must be used because the timing behaviour of a process will depend on the number of instances of an activity that are enabled.

Finally, the rate at which one PEPA component is able to evolve into another is defined. Given two PEPA components, P and P', such that P' is a one-step derivative of P, the *transition rate* between the two is denoted by $q(P, P')$. Thus the rate at which a component P may evolve into component P' is defined as the sum of the rates of the activities that relate the two i.e.

$$q(P, P') = \sum \{\!| \, \mathtt{r} \mid P \xrightarrow{(\alpha,r)} P' \, |\!\}$$

The underlying Markov process is isomorphic with the derivation graph of a PEPA model: these transition rates form the entries of the *infinitesimal generator matrix.* The derivation graph of the $System$ defined above is shown in Figure 1, where f defines the *apparent* rate of two synchronising activities (see [Hil94, Chapter 3]).

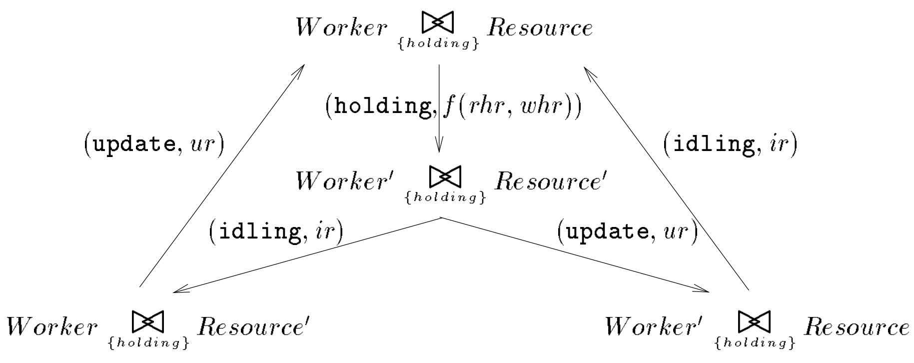

Figure 1: Derivation Graph of $System$

2.2 The PEPA Workbench

PEPA models of realistic systems are much more complex than the simple example shown above and tool support is vital to aid the generation and manipulation of the Markov model. To this end, the PEPA Workbench was developed [GH94]. It allows models to be defined using PEPA. These are then processed by the Workbench which creates a representation of the infinitesimal generator matrix. This is in a form suitable for analysis by several computer algebra, and especially linear algebra, packages. Thus the Markov process can be solved, and a steady state probability distribution calculated. However, at the moment, the Workbench does not support the automatic calculation of performance measures—this is done manually by the user.

3 Deriving Performance Measures

3.1 Reward Structures

Reward structures, as introduced by Howard [How71], provide a very general framework for specifying and deriving performance measures over continuous time semi-Markov processes. In its full generality, a reward structure consists of:

- a *yield* function, $y_{ij}(\sigma)$; while the process occupies state i of the semi-Markov process, having chosen a successor state j, it earns reward at a rate $y_{ij}(\sigma)$ at a time σ after entering the state.
- a *bonus* function, $b_{ij}(v)$; when the transition from state i to state j is made at some time v, the process earns the fixed reward $b_{ij}(v)$.

To derive steady state measures over a continuous time Markov process (which is generated from a PEPA model) rather than a semi-Markov process, rewards as general as these are not required. In such cases it suffices to fix the yield rate to be constant during the occupancy of a state, and to be independent of the successor state, i.e. $y_{ij}(\sigma) = y_i$. Bonus functions are discarded because information about transition rates is unavailable in the solved model. Therefore, a reward structure will consist of a constant yield function only. A measure is then calculated as the *total reward*:

$$\mathcal{R} = \sum_i y_i \cdot \pi_i$$

where π_i is the steady state probability of being in state i and the summation is taken over the complete state space; in the case of a PEPA model, the complete derivative set.

3.2 Deriving Performance Measures: the Current Approach

Currently the calculation of performance measures from PEPA models is ad hoc, and relies on the ingenuity of the modeller. In general a reward based approach is taken but there is no support for attaching rewards to states: the modeller must do it explicitly for each state, for each reward. For example, a modelling study might progress as outlined below.

1. Create a model and submit it to the PEPA Workbench for analysis. This will produce a representation of the state space, and the infinitesimal generator matrix. Submit the matrix data to a linear algebra package to generate the steady state distribution. This is all completely automated.

2. Form the reward structure over the state space which will generate the performance measure of interest. This involves deciding which states of the (perhaps huge) state space require consideration, and choosing values to be associated with each of these states. For instance, when calculating utilisation of a resource, the value 1 should be associated with each state in which use of the resource is "enabled", and the value 0 associated with all other states. This is not done automatically. Once a reward value is associated with every state a reward vector can be formed.

3. Use the steady state probability distribution to calculate the total reward. If a vector of the reward values for each state has been produced, this can be done automatically in whatever linear algebra package is being used.

Manually forming the reward structure over a complex model is time-consuming and error-prone. Moreover the process algebra style of model construction focuses on the system's behaviour rather than the individual states which the system passes through. The intention of the PEPA Workbench is to make this underlying, state-based representation of the system transparent to the modeller. However this objective cannot be achieved if the modeller must consider each individual state in order to derive any measures from the model.

The aim of our work has been to automate the second step of the above procedure and relieve the modeller of the responsibility of explicitly constructing the reward vector necessary for step 3. Moreover we do so by allowing the modeller to specify those states to which a reward should be attached in terms of their behaviour. In addition, we take advantage of the compositional structure of the model, allowing the specification to be defined on a particular component of interest, rather than on the whole model.

4 The PEPA Reward Language

Using the simple language described here, the process of specifying performance measures is split into two stages:

- Defining a *reward specification*, which associates a value with a particular process derivative, if it is capable of behaving as required.
- Defining an *attachment* which determines at which process derivatives a particular reward specification is evaluated.

Formally, each reward specification can be considered as a pair consisting of a logical formula and a reward expression. If the particular derivative under consideration is capable of the behaviour described in the logical formula, then a reward is assigned to that derivative. The reward corresponds to the evaluation of a simple arithmetic expression. The meaning of the reward specification will depend on how it is "attached" to a PEPA model. This is because the value assigned to a process derivative may depend on information local to the process derivative, for example the rate at which it can evolve.

Before defining the syntax of the rewards language, it will be necessary to describe the particular logic used in reward specifications.

4.1 Logic Syntax

The formalism chosen for defining properties of processes is based on a simple modal logic called Hennessy-Milner logic (HML, [HM85]). Formulae are built from Boolean connectives and modal operators; the syntax is given below.

$$\Phi \ (\mathsf{formula}) ::= \mathtt{tt} \ \mid \ \mathtt{ff} \ \mid \ \neg\Phi \ \mid \ \Phi_1 \wedge \Phi_2 \ \mid \ \Phi_1 \vee \Phi_2 \ \mid \ [K]\Phi \ \mid \ \langle K \rangle \Phi$$

K ranges over subsets of action *types*, that is $K \subseteq \mathcal{A}$. HML only allows a single action type to appear in a modality; thus this logic generalises HML.

However this logic is capable only of describing qualitative behaviour; there is no provision for specifying properties involving timing. Since, as described in Section 2, a PEPA process can be regarded as a labelled multi-transition system, formulae are interpreted with respect to this slightly more elaborate model.

4.2 Logic Semantics

The particular semantics chosen are simple and reflect the intended use of the logic in specifying simple behavioural properties of PEPA processes. A formula will be interpreted in conjunction with a particular state of the labelled multi-transition system. The evaluation of such a pair will result in a Boolean value. The notation for the evaluation of a formula Φ at a particular state s is $\| \Phi \|_s$, and the evaluation is defined inductively on the structure of Φ. The semantics are given in Figure 2.

$$
\begin{array}{rcl}
\| \texttt{tt} \|_s & = & T (\in \{T, F\}) \\
\| \texttt{ff} \|_s & = & F \\
\| \neg\Phi \|_s & = & \neg \| \Phi \|_s \\
\| \Phi_1 \vee \Phi_2 \|_s & = & \| \Phi_1 \|_s \vee \| \Phi_2 \|_s \\
\| \Phi_1 \wedge \Phi_2 \|_s & = & \| \Phi_1 \|_s \wedge \| \Phi_2 \|_s \\
\| [K]\Phi \|_s & = & \bigwedge \{ \| \Phi \|_{s'} \colon s \xrightarrow{(\alpha,r)} s', \alpha \in K \} \\
\| \langle K \rangle \Phi \|_s & = & \bigvee \{ \| \Phi \|_{s'} \colon s \xrightarrow{(\alpha,r)} s', \alpha \in K \}
\end{array}
$$

Figure 2: Semantics of the logic

No distinction has been made between Boolean operators and the operators of the modal logic—the context is sufficient for distinguishing which meaning is in use. The intuitive meaning of the modal operators is easily seen. $\| [K]\Phi \|_s$ is true if *for all* states s' such that there is a transition from s to s' via an activity whose type is in K, $\| \Phi \|_{s'}$ is true. Similarly, $\| \langle K \rangle \Phi \|_s$ is true if *there exists* a state s', such that there is a transition from s to s' via an activity whose type is in K, and $\| \Phi \|_{s'}$ is true. These operators do not make use of the fact that the semantics of PEPA generate a labelled *multi*-transition system. Since the logic only captures qualitative behaviour, the timing differences that result when states are joined by multiple identical activities are immaterial.

Next, reward expressions are described. In this paper, the syntactic class of alphanumeric sequences is denoted by string, and the syntactic class of expressions representing real numbers is denoted by real; these require no formal definition here.

4.3 Reward Expressions

The syntax of reward expressions is very simple, indeed it captures little more than a trivial syntax for arithmetic. The only additions to this are three bound variables. The syntax is given below :

$$\begin{array}{rcl} e\ (\mathsf{expr}) & ::= & (e) \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1/e_2 \mid \mathsf{atom} \\ \mathsf{atom} & ::= & \mathsf{real} \mid m_c \mid m_p \mid rate(\mathsf{action\text{-}type}) \end{array}$$

The bound variables, named m_c, m_p and $rate()$, will be used to denote real numbers. Their meaning will be dependent on the rewards already calculated, and the particular labelled multi-transition system which results from the PEPA model under consideration. They exist for pragmatic reasons—they are useful in specifying performance measures.

The variables m_c and m_p are intended to give the reward expression access to any previously assigned, and current rewards, allowing reward expressions to make use of a prior reward assignment made to a prior, or the current, derivative. This would allow a reward to be created by combining two simpler rewards. The function $rate()$, allows activity rates to be used in expressions—specifically, reward values can be assigned to a derivative P which make use of the transition rate from P to successor derivatives via an activity of type α.

Given the semantics for the logic, and a description of reward expressions, the PEPA reward language can now be described.

4.4 The Reward Language

The syntax of reward specifications is given below.

$$\begin{array}{rcl} \mathsf{reward} & ::= & \text{“reward”}\ \mathsf{string}\ \text{“=”}\ \mathsf{reward\_spec} \\ \mathsf{reward\_spec} & ::= & \mathsf{formula}\ \text{“}\Rightarrow\text{”}\ \mathsf{expr} \end{array}$$

A reward specification is identified by its name, and is a pair of a logical formula and a reward expression. A reward specification will be used to determine if a derivative is given a reward or not. If it is capable of the behaviour defined by the logical formula, it will be assigned as a reward the result of evaluating the reward expression. However, these reward specifications do not mean anything in isolation; a method of associating them with processes is required, as stated in Section 4. This requires an explanation of *attachments*; the syntax is given below.

attachment	::=	"attach" string "to" process modifiers
modifiers	::=	"follow" action-type

An attachment is a method of examining the derivative set of a PEPA process (or at a lower level, traversing the state space of the multi-transition system). It states which subcomponent is to be studied and hence which derivatives of that subcomponent will be considered for a reward. The idea is that starting at the initial subcomponent, the derivatives that can be reached *in the context of the complete model*, via activities of types defined in the attachment ("follow") set, will be considered for a reward. Therefore, the reward specification should be evaluated at a start point i.e. an initial PEPA process, and then the attachment used to determine to which derivatives the procedure should be reapplied. If P is the current derivative, $P \xrightarrow{(\alpha,r)} Q$, and α is in the attachment set, then in due course, Q will be considered for a reward too. Thus the reward assignment procedure is iterative. Of course, this vague description is less than satisfactory as a formal account, and the interested reader is referred to [Cla] for a full semantics.

5 Examples

In this section, the specification language is used to express some conventional performance measures. Two different models are used as vehicles for these examples; each is explained in turn.

5.1 Example 1

A model that occurs frequently in the literature is the *multi-processor multi-memory* system. This has the advantage that the process algebra style is a particularly suitable way to represent it. Consider a vast simplification of a "modern" computer system, containing several processors and several memory chips. Each processor performs some work, then attempts to access memory. We can model the individual components of this system in the following obvious way :

$$
\begin{aligned}
Mem_i &\stackrel{\text{def}}{=} (\texttt{get}_{\texttt{Mi}}, \top).(\texttt{rel}_{\texttt{Mi}}, \top).(\texttt{refresh}, r).Mem_i \\
Proc_j &\stackrel{\text{def}}{=} (\texttt{work}, w_j).\sum_k (\texttt{get}_{\texttt{Mk}}, g \cdot p_k).(\texttt{rel}_{\texttt{Mk}}, r).Proc_j \\
&\quad \text{where } \sum_k p_k = 1
\end{aligned}
$$

Each memory chip can be claimed and released (like a semaphore), but after being released, it performs a "refresh" to ensure the contents of the memory are maintained correctly. Each processor will perform some work, and will then choose which memory chip to access. Once finished, it will release it again. However, to make things more interesting, this particular multi-processor system will be extended with an "IO Controller" chip. This processor is only capable of accessing memory chip Mem_1 :

$$IOC \stackrel{\text{def}}{=} (\texttt{work}, w).(\texttt{get}_{\texttt{M1}}, g).(\texttt{rel}_{\texttt{M1}}, r).IOC$$

This can be combined with two processors and two memories to make a composite system :

$$
\begin{aligned}
System &\stackrel{\text{def}}{=} (Proc_1 \parallel Proc_2 \parallel IOC) \underset{L}{\bowtie} (Mem_1 \parallel Mem_2) \\
&\quad \text{where } L = \{\texttt{get}_{\texttt{Mi}}, \texttt{rel}_{\texttt{Mi}} \mid i = 1, 2\}
\end{aligned}
$$

Now it is possible to specify formally some interesting performance measures.

5.1.1 A Utilisation Measure

In order to determine how many memory chips are needed for reasonable performance, it may be interesting to determine what percentage of the time a particular memory chip is in use. In order to specify this, the behaviour that characterises the fact that the chip is in use is needed. This is easily seen—the chip is in use if it is capable of performing an activity of type $\texttt{rel}_{\texttt{M1}}$; if so, it previously performed a $\texttt{get}_{\texttt{M1}}$ in synchrony with a processor which required access. The required reward specification is then :

$$\texttt{reward util-mem1} = \langle \texttt{rel}_{\texttt{M1}} \rangle \texttt{tt} \Rightarrow 1$$

with an appropriate attachment being :

$$\texttt{attach util-mem1 to } System \texttt{ follow} -$$

It is worth explaining in a little detail why this will achieve the desired result. The first thing to note is that the attachment is very simple—rather than examining any particular subcomponent of the model, the whole system is analysed. This does no harm in this instance, nothing more focused is required. The reward assignment procedure will begin with the process $System$, and will determine whether it is capable of satisfying the logical formula $\langle \mathtt{rel_{M1}} \rangle \mathtt{tt}$. Clearly none of the memory chips are in use at this point, so it will not be possible to release any, and specifically not Mem_1. Thus the formula is false here, and the reward of 1 is not assigned. The next stage is to take the one-step derivatives of $System$ via the activities of type defined in the attachment. Any action type was allowed, so each one-step derivative is examined in similar fashion to $System$. This is done exhaustively until no further derivatives exist that have not been tested for a reward. At this point the procedure will terminate. Therefore any derivative of $System$ that could perform an activity of type $\mathtt{rel_{M1}}$ will be given the reward 1. With the assumption that other derivatives have a reward of 0, this procedure will produce an appropriate vector, which when combined with the steady state probability vector, will produce a value between 0 and 1. This of course can be interpreted as the percentage of time that Mem_1 is in use.

The above example would become slightly more complicated if the percentage of time a bona fide processor was waiting to access memory was required (thus ruling out the IO Controller chip). However this is still simply achieved. Behaviourally, a processor is waiting for access to memory if it is waiting for access to *any* of the memory chips (unlike IOC which may only be waiting to access Mem_1). Therefore, an appropriate reward specification would be :

$$\mathtt{reward\ proc\text{-}waiting} = \langle \mathtt{get_{M1}} \rangle \mathtt{tt} \wedge \langle \mathtt{get_{M2}} \rangle \mathtt{tt} \Rightarrow 1$$

since it should be able to use Mem_1 or Mem_2, whichever should become available. The same attachment described above is suitable here.

5.1.2 A Throughput Measure

Another useful performance measure may be the throughput of IO Controller accesses to memory. Here it is possible to take advantage of the forced-flow law which states that the throughputs in all parts of a system must be proportional to each other. Since the IO controller is a sequential process, its throughput is determined by either of its possible activities. Arbitrarily choosing one leads to the following reward :

$$\mathtt{reward\ IO\text{-}throughput} = \langle \mathtt{work} \rangle \mathtt{tt} \Rightarrow rate(\mathtt{work})$$

If the derivative can perform an activity of type `work`, then it will be assigned the value of $rate(\texttt{work})$. This will evaluate to the sum of the rates at which the derivative can evolve via an activity of type `work`. Therefore the value will give a measure of the throughput of the `work` activity, and, by the forced-flow law, of the whole system. However this time, attention is restricted to the IO controller subcomponent, IOC; the attachment is :

$$\texttt{attach IO-throughput to } IOC \texttt{ follow} -$$

The difference in the assignment procedure is that the logical formula is only checked against activities of type `work` *that can be performed by IOC in the context of System*. This means that if the IO Controller is in a position to perform a `work` activity, but its context does not allow it (e.g. it must synchronise but no other process is willing), then the activity is impossible. Restricting attention to the subcomponent in this way ensures that if by chance any other processes may perform an activity of type `work`, it will not influence the reward assigned. This time the reward vector will contain activity rates, and when combined with the steady-state vector will give the throughput as required.

5.1.3 Illustrating the Logic

In order to illustrate the discriminatory power of the modal logic, a more contrived example is given next. Suppose it was useful to know the percentage of time only one memory chip was in use (and thus not both). First note that both memories are in use if a particular derivative enables a $\texttt{rel}_{\texttt{Mi}}$ activity, and its one-step derivative via $\texttt{rel}_{\texttt{Mi}}$ enables a $\texttt{rel}_{\texttt{Mj}}$ activity. Combining this with the example above, the following reward specification is obtained :

$$\begin{aligned}\texttt{reward one-mem} &= (\langle\texttt{rel}_{\texttt{M1}}\rangle\texttt{tt} \vee \langle\texttt{rel}_{\texttt{M2}}\rangle\texttt{tt}) \\ &\wedge [\texttt{rel}_{\texttt{M1}}][\texttt{rel}_{\texttt{M2}}]\texttt{ff} \\ &\wedge [\texttt{rel}_{\texttt{M2}}][\texttt{rel}_{\texttt{M1}}]\texttt{ff} \Rightarrow 1\end{aligned}$$

This can be attached to the whole system in the naïve manner of the first example, and will only assign a reward to those derivatives that allow the release of either memory chip, *but not both*. Therefore these derivatives correspond to states in which one memory chip is in use only.

5.2 Example 2

To motivate the second example, another system is introduced. Consider the PEPA model below :

$$
\begin{aligned}
Buffer_0 &\stackrel{\mathrm{def}}{=} (\mathtt{packet}, p).Buffer'_0 \\
Buffer_i &\stackrel{\mathrm{def}}{=} (\mathtt{packet}, p).Buffer'_i + (\mathtt{discard}, d).Buffer_{i-1} \\
Buffer'_i &\stackrel{\mathrm{def}}{=} (\mathtt{discard}, d).Buffer_i + (\mathtt{verify}, v).Buffer_{i+1} \\
&\ldots
\end{aligned}
$$

This models a buffer which accepts possibly corrupt packets from a source. It abstracts from the difference between discarding a packet due to corruption and removing a packet from the buffer. Frequently it is useful to know a measure such as the average number of packets in the buffer. This can be done by basing the reward assigned to a derivative on the reward assigned to a previous derivative. For example, a particular derivative may correspond to a buffer containing n packets. By a particular sequence of activities, this could evolve into a derivative that represents a buffer with $n + 1$ packets. Clearly, the reward assigned to the second derivative could be the value of the reward assigned to the first derivative, plus one. A sensible reward specification would be :

$$\mathtt{reward\ count} = \langle\mathtt{discard}\rangle\mathtt{tt} \Rightarrow m_p + 1$$

This states that at any point where the buffer is able to discard another packet, it contains one more packet than it did before its last activity (which would correspond to the input of the packet). A possible attachment could be :

$$\mathtt{attach\ count\ to}\ Buffer_0\ \mathtt{follow}\ -$$

However, this will not have the intended effect since both $Buffer'_i$ and $Buffer_{i+1}$ may perform the `discard` activity. The solution is to notice that a packet is only truly input after a `verify` activity, and so the attachment should be changed thus :

$$\mathtt{attach\ count\ to}\ Buffer_0\ \mathtt{follow\ verify}$$

6 Conclusions and Future Work

By integrating a modal logic, complementary to a process algebra, and a reward structure, complementary to a Markov process, we have defined the PEPA reward language which allows the modeller to specify performance measures in terms of model behaviour. Moreover, with the prototype implementation, this approach can be used to support modelling studies conducted using the PEPA Workbench.

There are several possible directions for future work :

- Considering transient analysis and performability measures would introduce the need for bonus functions as well as yield functions in the reward structure.
- Since an expression in the PEPA reward language can be used to partition the state space of the underlying model we hope to investigate the use of such expressions to perform state space truncation leading to approximate solutions.
- Including a predefined set of measures over components, within the PEPA Workbench, in the form of parameterised expressions in the PEPA reward language would allow an even more abstract specification of performance measures. For example a modeller would not have to consider activities at all, but could rather ask questions such as "what is the throughput of this processor?"

Acknowledgements

We would like to thank Gordon Brebner for helpful comments on a draft of this paper. Graham Clark is supported by EPSRC grant 95306547.

References

[Cla] G. Clark. Formalising the Specification of Rewards with PEPA. To appear in Proceedings of PAPM 1996.

[GH94] S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In G. Haring and G. Kotsis, editors, *Proceedings of 7th Conf. on Mod. Techniques and Tools for Computer Perf. Eval.*, volume 794 of *LNCS*, pages 353–368, 1994.

[GHR93] N. Götz, U. Herzog, and M. Rettelbach. Multiprocessor and Distributed System Design: The Integration of Functional Specification and Performance Analysis using Stochastic Process Algebras. In *Performance'93*, 1993.

[Her] H. Hermanns. Performance Prediction of Behavioural Descriptions with Temporal Logics. Extended Abstract.

[Her90] U. Herzog. Formal Description, Time and Performance Analysis: A Framework. Technical Report 15/90, IMMD VII, Friedrich-Alexander-Universität, Erlangen-Nürnberg, Germany, September 1990.

[Hil94] J. Hillston. *A Compositional Approach to Performance Modelling.* PhD thesis, The University of Edinburgh, 1994. CST-94-108.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, January 1985.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, Englewood Cliffs, NJ, 1985.

[How71] R. A. Howard. *Dynamic Probabilistic Systems*, volume II: Semi-Markov and Decision Processes, chapter 13, pages 851–915. John Wiley & Sons, New York, 1971.

[Mil89] R. Milner. *Communication and Concurrency.* International Series in Computer Science. Prentice Hall, 2nd edition, 1989.

[SW89] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89:333–354, 1989.