Techniques for the Construction and Analysis of Algebraic Performance Models

Graham Clark

Doctor of Philosophy
University of Edinburgh
2000

Abstract

The performance modeller may attempt to quantitatively analyse the behaviour of computer systems by building performance models. Such models may become unwieldy, and so high-level structured modelling techniques have been developed. A *stochastic process algebra* (SPA) provides such a technique, a compositional modelling calculus. Hillston's PEPA is an SPA, a classical process algebra enhanced to represent the performance of systems. This thesis uses PEPA as a foundation, and examines different ways to assist the SPA performance modeller.

A weak stage in the SPA methodology is the calculation of concrete performance measures, since much research does not focus beyond a steady-state probability vector. A framework is developed for specifying steady-state performance measures for PEPA models. The technique is used at the high-level of the process algebra, and not applied directly to states, or the stochastic process. It employs an enhanced modal logic to allow the modeller to identify interesting model behaviour. Furthermore, the modeller may choose to study only the behaviour of subcomponents in the model context. The method automatically specifies a Markov reward model (MRM). The modal logic is suitably expressive; it is shown to characterise PEPA's *strong equivalence* relation. Conditions are given under which model subcomponents may be aggregated such that the MRM is guaranteed to be *strongly lumpable*. The technique is compared to various other solutions to the reward specification problem.

If a randomly distributed model feature possesses the *insensitivity* property, then the equilibrium solution of the model only depends on the mean of the distribution. A new SPA combinator is defined which builds a model from a set of simple components restricted to *queue* to perform particular activities. It is demonstrated that a subset of the activities of these SPA models are insensitive, and therefore may have generally distributed durations. Furthermore, it is proven that a model of this structure exhibits a *product form* solution over its submodels, allowing its solution to be expressed as a product over the smaller solutions of its parts. This work leads to a more general examination of insensitivity in SPA models. An extension to PEPA is defined which allows activities with *generally distributed* durations. Balance conditions are given which guarantee the insensitivity of these activities. However these conditions are strong, and at the level of the stochastic process.

Models of *transaction processing systems* (TPS) are presented as a case study. These systems consist of a centralised database, and a set of *transactions* which access database objects. Sample performance measures are specified for TPS models, and a model of a TPS is constructed using the new combinator, guaranteeing the insensitivity of a subset of its activities.

Declaration

Background on the PEPA modelling tools described in Chapter 2 appeared in [18]. Initial work which led to the research presented in Chapter 3 was published as [15, 19] and contributed to [17]. The results presented in Chapter 4 are published in [16].

I hereby declare that the research recorded in this thesis and the thesis itself was composed and originated entirely by myself, except where detailed above, in the Laboratory for Foundations of Computer Science at The University of Edinburgh.

Graham Clark

Acknowledgements

Foremost I thank my supervisor, Jane Hillston, for her support, guidance, and not least patience. I have always been struggling to reach her high standards, and without her determination on my behalf, this thesis would not have been written. Thanks also to Stephen Gilmore for thorough proof-reading.

I am most grateful to my friends for their enduring company. In particular, I mention Andrew and Steven; my flatmates Marcus, George and James; and my officemates Álvaro and Mark. Thanks to my family for support during my postgraduate years. This thesis was supported by EPSRC award 95306547.

Finally, I would like to pay my respects to Ian Tweedie. We became friends at the time I began my postgraduate work, and Ian was my climbing partner for one and a half years before he moved to London to pursue a career as a technology consultant. In May 1999, Ian died suddenly of heart failure, caused by a condition called Cardiomyopathy. He is missed by all his friends. More on Ian's life can be found at `http://go.to/Ian_Tweedie`.

Table of Contents

Chapter 1

Introduction

1.1 Background and Aims

The work presented in this thesis adds to the repertoire of techniques available to the stochastic process algebra (SPA) performance modeller. The thesis is concerned with both the construction, and the performance analysis, of SPA models. Performance modelling deals with the analysis of the dynamic behaviour of systems. Models of such systems can quickly become large and unwieldy, and so building these models in a compositional fashion can greatly assist analysis. This motivates the choice of PEPA [44], a prominent stochastic process algebra, as the modelling language for the work in this thesis. This chapter presents an overview of the thesis, identifying the main results.

PEPA is also termed a Markovian process algebra (MPA), meaning that the performance aspects of a PEPA model can be understood by a translation to a (continuous time) Markov chain. The stochastic process can then be mathematically exploited, for example, to calculate the steady-state probability distribution over the states of the model. In the literature, the Markov chain has been extended specifically to cater for the extraction of performance properties. A Markov reward model (MRM), equips a Markov chain with a reward structure (or reward function), a map over the states of the stochastic process to the real numbers. Different reward structures can be used to analyse classes of model measures (depending on interpretation), for example steady-state, transient and cumulative measures. Furthermore, such reward structures have been employed in the analysis of several classes of modelling languages, for example stochastic Petri nets (SPN) and queueing models [36]. Methods also exist for specifying a reward structure over a SPA model, in order to calculate steady-state measures (for example [5]). One aim of this thesis is to provide a powerful technique for a PEPA modeller to specify a reward structure for steady-state measures. The method exploits the compositional benefits of working with a process algebra, and allows performance measures to be automatically calculated. In order to improve the uptake of process algebras, the modeller must have intuitive, useful and automated methods to go beyond a steady-state probability distribution, to the calculation of numerical results. The PEPA Reward language employs a modification of a published modal logic. Process algebras and modal logic enjoy a fruitful relationship, in that they both can be defined in terms of the same mathematical structures. The modeller uses the logic to identify model states with a required property, and may then assign a particular reward to these states. Implicitly, this builds a MRM. The theory of PEPA defines several equivalence relations, the most useful of which corresponds to lumpability over the Markov chain of a model. Called strong equivalence, it serves as the basis for an exact model aggregation procedure, thus serving to reduce the size of the performance models. The connection between the modal logic of the PEPA Reward language and a PEPA model give the modeller conditions to ensure that rewards assigned to strongly equivalent states are equal. This means that the modeller can apply a model aggregation procedure without having to alter the specification of any rewards.

Recently, much research effort has been spent on extending the expressiveness of stochastic process algebra. Continuous time Markov chains are the subject of a wealth of analysis techniques, but it has been argued that in many real-life circumstances, the Markovian assumption is unrealistic. Markovian process algebra models feature timed activities which are exponentially distributed only. PEPA is an MPA, and employs an interleaving semantics. In essence, this means that in building the stochastic process, the independent parallel composition of two activities is treated as a choice between the two possible sequential orderings. When activity durations are distributed according to a negative exponential random variable, this interpretation is valid, since when one activity completes, the time the other has been active can, statistically, be disregarded—this is why the negative exponential distribution is termed memoryless. However, many phenomena are not characterised by exponential distributions, for example timeout features in network protocols. This has led to several new process algebra designs, with so-called 'start-termination' (ST) semantics, where an activity is split into two atomic events—enabling, and completion. Models of these algebras are understood by a translation to a more flexible stochastic process, the generalised semi-Markov process (GSMP). States of this process are characterised by sets of active elements, where the duration of each element can be distributed arbitrarily. These elements are given lifetimes sampled from their distributions, and then notionally time passes until a lifetime completes. The completing element and the current state determine the successor state; on changing state, elements which are also present in the successor state retain their spent lifetimes. This property makes the GSMP suitable as a model for a generally distributed SPA, but unfortunately, analysis is difficult in general. A common approach is to resort to simulation in order to determine performance measures.

In this thesis, the GSMP is also employed, but the motivation of the work is different—it is to establish conditions in MPA (specifically PEPA) models where the modeller *need not assume* that activities are exponentially distributed. This work makes use of the theory of insensitivity in stochastic processes, a theory which was developed in the context of the GSMP. If a GSMP is insensitive to its generally distributed active elements, then these active elements may be arbitrarily distributed—with identical means—and the steady-state solution is provably equal. Insensitivity of a process can be guaranteed by the satisfaction of a set of insensitivity balance equations over that process. The theory of insensitivity has been applied to other high-level modelling paradigms, for example stochastic Petri nets [38]. The motivation of this work was to investigate the introduction of generally distributed *transitions* in models, and applications include aggregating particular places of the net to give an insensitive 'skeleton' structure, and incorporating *age dependent* routing, where the probability of choosing a successor marking depends on the time at which a transition fires. However, such work on Petri nets seems to be hampered by the lack of structure in models, and it seems to be unclear how such techniques could be applied systematically to more complex Petri net models.

Insensitivity theory suggests a route by which generally distributed activities can be incorporated into MPA models. This is done by providing a faithful mapping from a particular structure of MPA model to a GSMP, where activities are represented in the active elements of the process. By showing the insensitivity balance equations are satisfied, it can be deduced that the activities of the MPA model may be arbitrarily distributed with the same mean. This approach differs from others, in that it exploits a particular structure of PEPA model, and guarantees that all models built to this structure will possess the given insensitivity property. Such work is related to published research on identifying MPA models with a *product form* solution. A PEPA model has a product form solution if the solution of the model can be expressed as a product term where the solutions of the model subcomponents are present as subterms. For example, one structure of model [46] exhibits several independent components which compete for access to a resource, and has been shown to exhibit product form. The insensitive structure identified in this thesis can also be viewed as a contention between subcomponents over resources, where subcomponents may be blocked and forced to queue in order to proceed. Interestingly, the structure designed to be insensitive to generally distributed activities also possesses a product form solution over subcomponents.

A result by Hillston [43] showed that under some conditions on the cooperation present in a PEPA model, a generally distributed activity could be safely introduced. This alternative approach motivates the further study of insensitivity later in this thesis. Instead of relying on the GSMP model, general distributions can instead be arbitrarily closely approximated by the use of a distribution called the Erlang mixture. Introducing Erlang mixtures into PEPA models ensures the model remains Markovian, but also means that it can become infinite-state. However, this is shown to happen in a controlled fashion, such that the analysis of these models remains possible. These models incorporating generally distributed activities are not built according to some structural recipe, but they must satisfy some conditions. For example it must be the case that two generally distributed activities cannot both become enabled simultaneously. From here, a balance equation property similar in spirit to insensitivity balance, guarantees that the steady-state solution of each of these models is identical to that possessed by the model's exponentially distributed counterpart.

These two strands extend the range of techniques available to construct and to analyse SPA models. With the uptake of process algebras on the rise, it is important to show that these techniques are useful in practice. Models of *transaction processing systems* are used to showcase the research presented in this thesis.

1.2 Synopsis

In this section, a more detailed chapter-by-chapter breakdown of the thesis is presented.

Chapter 2 presents the mathematical background to the thesis. This draws on a variety of notation commonly employed in the study of SPA, as well as giving a brief introduction to the concepts employed from the theory of stochastic processes. PEPA, the SPA used for the presentation of this thesis work, is introduced, including a description of the key concepts required for this thesis. Amongst these are the operational semantics, the translation to a continuous time Markov chain, and the strong equivalence relation. A breakdown of types of performance measures and reward structures used for stochastic processes is presented, and finally, the tools available for the analysis of MPA models are discussed.

Chapter 3 focuses on the analysis of PEPA models. The PEPA Reward language is motivated as a means to specify and automatically generate performance measures from PEPA models. The modal logic PML_μ is introduced, and is shown to characterise PEPA's strong equivalence relation, in that two strongly equivalent PEPA processes are shown to satisfy exactly the same PML_μ formulas. Some results on SPA *contexts* are then presented. This will provide the necessary theory for allowing the study of the behaviour of PEPA subcomponents in the context of a larger model. The definition of the satisfaction of PML$_\mu$ formula in context is provided, and conditions are given for the safe aggregation of subcomponents in context such that the truth of any PML$_\mu$ formula is preserved. The PEPA Reward language is then formally defined, making use of PML$_\mu$ and PEPA contexts, and some examples given to illustrate its use.

Chapter 4 introduces the theory of insensitivity, and the generalised semi-Markov process on which it was based. This motivates the study of insensitivity in PEPA models. A structure of model is devised, the form of which can be built by the application of a derived PEPA combinator to a set of sequential components, that is components without the PEPA cooperation combinator. The derived combinator implicitly introduces an *arbiter* subcomponent, which is placed in cooperation with the sequential components. The effect of the arbiter is to introduce bottlenecks, such that at particular points, the subcomponents must queue, and all those not at the head of the queue are blocked. A mapping from this structure of model to a GSMP is then exhibited, such that the GSMP preserves the performance semantics of the PEPA model, and where the active elements of the GSMP are built using PEPA activities. A general solution form for these models is demonstrated, and shown to be a product form over the sequential components of the queueing combinator. A set of insensitivity balance equations is then formed, and shown to be consistent with the general form of solution. This implies that the model is insensitive to its residence time in those substates (states of each subcomponent) where the subcomponent is not either currently entering or leaving a queue. In particular, this implies that for these substates, an enabled activity may be generally distributed.

In Chapter 5, insensitivity is developed by modelling general distributions with the Erlang mixture. This distribution is a probabilistic sum over sequences of exponential distributions, and can approximate many distributions arbitrarily closely. gPEPA is introduced, a syntax for describing PEPA models with generally distributed activities. Several restrictions are placed on gPEPA; significantly, two generally distributed activities may not synchronise, and furthermore, two may not be newly enabled simultaneously. A semantics for gPEPA is presented which generates a transition system built with both probabilistic and exponential transitions. The transition system is shown to be consistent, and a (possibly) infinite-state continuous time Markov chain model is developed. From this, more balance equations are formulated, which specify that the probabilistic flux out of a sequential component of the model is equivalent to the flux into that component. It is shown that for a given gPEPA model, if sequential local balance holds for all sequential components enabling generally distributed activities, then the model is insensitive to these activities. This generalises a result originally presented by Hillston [43].

These chapters demonstrate the two themes of this thesis—providing techniques for the construction and the analysis of PEPA models. Chapter 6 presents a case study which exemplifies these techniques. Models of a *transaction processing system* are developed in PEPA. The first model is built using the combinator presented in Chapter 4, and therefore it is shown that it is unnecesary to assume that parts of the model need be exponentially distributed. Subsequently, models based on the work of Pun and Belford [63] are presented, and the PEPA Reward language is used to automatically generate performance measures consistent with those demonstrated in [63].

Chapter 7 concludes the thesis, and proposes directions for future work.

Chapter 2

Background

2.1 Introduction

This chapter presents the background material necessary for this thesis. It begins with some mathematical preliminaries vital for much of the work presented, including some material on random variables, probability theory and stochastic processes. Then PEPA is introduced, the stochastic process algebra used to present the major work in this thesis. PEPA is briefly related to other stochastic process algebras, and its key features are discussed, such as the operational semantics, its interpretation as a continuous time Markov chain, and the equivalence relations with which it is equipped. The tools available for working with PEPA models are then described. Finally, some background more specific to the work in the main chapters is summarised—firstly, the use of reward structures, and secondly, the use of general distributions, in high-level performance modelling.

2.2 Mathematical Preliminaries

2.2.1 Some Notation

Throughout this thesis, the following notation is used consistently. A vector of n items is denoted $\langle v_1, v_2, \ldots, v_n \rangle$ and is represented succinctly by $\underline{v}$. Concatenation of vectors is denoted by their juxtaposition, for example $\underline{uv}$. (x, y) represents an ordered pair, and a multiset S of items v_i is formally a set of pairs (v_i, j), where each v_i represents an item, and $j > 0$ the number of occurrences of that item in the multiset. The multiset is denoted $\{\!|v_1, v_2, \ldots|\!\}$. Set membership notation, $v_i \in S$ is abused to mean $(v_i, j) \in S$ for some j when the multiset is clear from context. $\uplus$ is used to represent multiset union, and is defined as traditionally. $\mathbf{1}(.)$ is an *indicator* function which takes a predicate as an argument, such that $\mathbf{1}(P) = 1$ if P is true, and $\mathbf{1}(P) = 0$ otherwise.

2.2.2 Probability

Probability is important in modelling many real-life systems, including computer and communications systems. Probability theory plays an integral part in the theory of stochastic process algebras. In this section, some notation and commonly used results are presented.

Consider a *sample space*, Ω, representing the set of all possible outcomes of an experiment. An observable outcome of an experiment is formalised by an *event*, a subset of Ω. An *event space* satisfies the following properties:

Definition 2.2.1. *An event space ε over a sample space Ω is a subset of 2^Ω such that*

- $\Omega \in \varepsilon$*—this event occurs for every outcome of the experiment.*
- *If $e \in \varepsilon$ then $\overline{e} \in \varepsilon$—if e denotes the occurence of an observable, then $\overline{e}$ denotes the fact that the observable did not occur.*
- *If $e_i \in \varepsilon$ for $i \geq 1$ then $\bigcup_{i=1}^{\infty} e_i \in \varepsilon$—this means if any of e_i can individually occur then 'one of e_i occurred' is also an event.*

A space satisfying these axioms is also known as a *sigma algebra*. A *probability measure* P is defined over an event space, the set of observable outcomes of an experiment.

Definition 2.2.2. $\Pr(.)$ *is a probability measure on an event space ε with sample space Ω if the following axioms hold:*

- $0 \leq \Pr(e) \leq 1$ *for all* $e \in \varepsilon$
- $\Pr(\Omega) = 1$
- $\Pr(\bigcup_{i=1}^{\infty} e_i) = \sum_{i=1}^{\infty} \Pr(e_i)$ *for* $e_i \in \varepsilon$, $e_i \cap e_j = \emptyset$ *if* $i \neq j$

Therefore, all probabilities lie in the range $[0, 1]$, and the probability that the outcome of any experiment lies in the entire sample space is 1, that is, it is certain. The probability that either of two mutually exclusive events occur is given by the sum of their respective probabilities.

When determining the probability of an event e, there may be information about a related event f, which has occurred. Therefore the probability of e is *conditioned* on f, that is the sample space of e is restricted to f. The probability of e conditional on f is given by $\Pr(e \cap f)/\Pr(f)$ and is denoted in this thesis by $\Pr(e \mid f)$.

2.2.3 Random Variables

Random variables play an important rôle in this thesis. PEPA models implicitly employ exponentially distributed random variables, and later work in this thesis attempts to generalise PEPA by introducing generally distributed random variables into the algebra. Some notation and preliminary definitions are given here.

Definition 2.2.3. *A random variable X is a function from a sample space Ω to the real numbers. The probability that X is in a subset R of the real numbers is given by* $\Pr(X \in R) = \Pr(A)$ *where $X(a) \in R$ if and only if $a \in A$ and A is an event.*

Therefore the probability assigned to a random variable is the probability of its inverse image in the sample space, provided that this is an event. In this thesis, X and Y will be used to range over random variables.

The probabilistic characteristics of a random variable can be defined entirely in terms of a *cumulative distribution function* (CDF, also *probability distribution function*). Given a random variable X, the CDF is denoted $F_X(\cdot)$, and is given by $F_X(y) = \Pr(X \leq y)$, meaning the probability that the random variable X takes some value less than or equal to y. The event space consists of the sigma algebra over subsets of the real line of the form $(a, b]$, that is open on the left and closed on the right. This implies that a CDF is non-decreasing, right continuous, and that $\lim_{x\to\infty} F(x) = 1$ and $\lim_{x\to-\infty} F(x) = 0$. All work in this thesis is with *continuous* random variables, that is where the range of values taken by each random variable is uncountable.

The support set of a random variable X is the closure of the set of arguments a of X for which $X(a) \neq 0$. Each random variable used for modelling in this thesis can be viewed as a positive 'time-to-complete' of some model event. This implies that for each such random variable X, its support set S_X will be such that if $a < 0$ then $a \notin S_X$. The *hazard rate* of a random variable, $h_X(y)$, is the probability per unit time that the random variable will take a value in an arbitrarily small range after y given that it does not take a value in $[0, y]$. More precisely, this is given by a limit statement, i.e.

$$h_X(y) = \lim_{\Delta y \to 0} \frac{\Pr(X \in (y, y + \Delta y) \mid X > y)}{\Delta y} \tag{2.2.1}$$

Markovian performance modelling makes exclusive use of the *negative exponential* random variable.

Definition 2.2.4. *A random variable X is* negatively exponentially *distributed with parameter λ if*

$$F_X(y) = 1 - e^{-\lambda y}$$

for all $y \geq 0$*, and* $F_X(y) = 0$ *otherwise.*

A common abbreviation for negatively exponentially distributed is exponentially distributed; these terms have the same meaning. The *expected value* of X with parameter λ is given by

$$\begin{aligned} E[X] &= \int_{-\infty}^{\infty} y dF_X(y) \\ &= \left[-ye^{-\lambda y}\right]_0^{\infty} + \int_0^{\infty} e^{-\lambda y} \\ &= 0 + \left[-(1/\lambda)e^{-\lambda y}\right]_0^{\infty} = 1/\lambda \end{aligned} \tag{2.2.2}$$

The hazard rate, or *failure* or *completion rate* of an exponential random variable X with parameter λ is given by

$$\begin{aligned} h_X(y) &= \lim_{\Delta y \to 0} \frac{((\lambda e^{-\lambda y}\Delta y)/(1 - F_X(y)))}{\Delta y} \\ &= \lim_{\Delta y \to 0} \frac{((\lambda e^{-\lambda y}\Delta y)/e^{-\lambda y})}{\Delta y} \\ &= \lambda \end{aligned} \tag{2.2.3}$$

In fact an exponential random variable is characterised by having a *constant* rate of failure, or completion. The exponential random variable is said to be *memoryless*; this means that the probability that it completes after some time t is the same as the probability it completes after some time $t + s$ given that it has not completed by s. This property is very convenient in giving a stochastic semantics to PEPA processes, as detailed in Section 2.3.3. In this thesis, an exponential random variable with parameter λ is denoted $\mathsf{Exp}(\lambda)$.

Many other kinds of continuous random variable are useful in performance modelling. For example, the uniform random variable has two parameters, a and b, $a < b$, and defines a random variable where the probability of any value varies uniformly between a and b i.e.

$$\Pr(X \leq y) = \begin{cases} (y-a)/(b-a) & \text{if } y \in [a, b] \\ 0 & \text{if } y < a \\ 1 & \text{otherwise} \end{cases}$$

A uniformly distributed random variable will be denoted $\mathsf{Uni}(a, b)$. Random variables may also be formed by combining exponential random variables. For example, the *Erlang-k* random variable consists of the *convolution* of k exponential random variables. This means that the probability that an Erlang-k distribution takes a value less than or equal to y is given by the probability that k identically distributed exponential random variables take values that when added together are less than or equal to y. An Erlang-k distribution models the time a customer takes to pass through k identically distributed exponential service centres in a queueing network.

Some common operations on random variables are used in this thesis. For example, assuming X and Y are independent, the sum of the two, $Z = X + Y$ has a distribution function given by

$$F_Z(t) = \Pr(X + Y \leq t) = \int_0^\infty F_X(s-t)dF_Y(s) \tag{2.2.4}$$

In similar fashion, the difference of two random variables, $Z = X - Y$, can be taken—the result is the random variable such that when it is convolved with Y gives a distribution function equal to $F_X(\cdot)$. The minimum of (independent) X and Y is the random variable that intuitively characterises the winner of a 'race' between X and Y. It is denoted by $\mathsf{min}(X, Y)$, and the distribution function is given by

$$F_{\mathsf{min}(X,Y)}(t) = 1 - \Pr(X > t, Y > t) = 1 - \int_0^\infty (1 - F_X(t))(1 - F_Y(t))dt \tag{2.2.5}$$

More generally, the minimum of a set of independent random variables, $S = \{X_i\}$, is denoted by $\mathsf{min}\{X_i \in S\}$. Finally, the probability that X is less than Y is defined as

$$\Pr(X < Y) = \int_0^\infty (1 - F_Y(t))dF_X(t) \tag{2.2.6}$$

Intuitively, this gives the sum over each infinitesimally small interval of the probability that X takes a value in that interval, while Y takes a value to the right of that interval. In the case that X and Y are exponentially distributed, some of these expressions can be analytically simplified. Let the parameters be λ and μ respectively. Then

$$\begin{aligned} \Pr(X < Y) &= \lambda/(\lambda + \mu) \\ F_{\mathsf{min}(X,Y)}(t) &= 1 - e^{-(\lambda+\mu)t} \end{aligned} \tag{2.2.7}$$

Therefore, the minimum of two exponentially distributed random variables is also exponentially distributed—the exponential is closed under minimum.

2.2.4 Stochastic Processes

A stochastic process provides a mathematical description of an evolving system, describing the sequence of states it enters over time. Formally, a stochastic process is defined to be a family of random variables, $\{X_t : t \in T\}$. T denotes the parameter space, and the random variables represent the measurements of some physical characteristic of the system, parameterised by $t \in T$. The domain of each X_t is given by some set S, the state space of the process. The parameter space is often taken to represent time; therefore any set of instances of $\{X_t : t \in T\}$ can be viewed as a sample path of the process, its position given by X_t at time t.

When T is countable, the stochastic process is called *discrete time*; otherwise the process is said to be *continuous time*. When studying a stochastic process, a modeller may typically wish to ascertain properties such as *first passage time*, that is the distribution of the time taken to reach state $B \in S$ given that the current position is state $A \in S$. Alternatively, a common requirement is the probability of finding the process in some subset of S in the 'long term', that is as $t \to \infty$. This latter property is of crucial interest when modelling with PEPA, and in this thesis. A state i of a process is said to be *recurrent* if it is certain (the probability is 1) that the first passage time of i, starting at i, is not infinite. It is *positive-recurrent* if the expected value of the first passage time is not infinite. A stochastic process is said to be *stationary* if the distribution of $X(t+s) - X(t)$ is independent of t. A process is said to have the *Markov property* (and therefore be a *Markov process*) if for $t_1 < t_2 < ... < t_n < t \in T$,

$$\Pr(X_t = x \mid X_{t_1} = x_1, \ldots, X_{t_n} = x_n) = \Pr(X_t = x \mid X_{t_n} = x_n) \tag{2.2.8}$$

That is, given the value of X_t at some $t \in T$, the future path of X_s for $s > t$ does not depend on knowledge of X_u for $u < t$; the current state is the extent of the memory of the process. A *continuous time* Markov chain is defined to be a stochastic process possessing the Markov property, and with a continuous parameter space T. Later in this chapter, it will be shown that PEPA can be given a stochastic process semantics in terms of a *continuous time Markov chain.* In fact all stochastic processes used in this thesis are continuous time.

Suppose a continuous time Markov chain is in a state $i \in S$ at time t. The time that the process will remain in state i before making a transition to state $j \neq i$ is exponentially distributed with parameter denoted q_{ij}. Therefore at time $t + \Delta t$, the system will be in state $j \in S$ with probability

$$\begin{array}{ll} q_{ij}\Delta t + o(\Delta t) & \text{if } i \neq j \\ 1 - \sum_{j \neq i} q_{ij}\Delta t + o(\Delta t) & \text{if } i = j \end{array}$$

If the Markov process has n states, it is characterised by its $n \times n$ *infinitesimal generator matrix* $Q = (q_{ij})$, where $q_{ii} = -\sum_{j \neq i} q_{ij}$ by convention.

Since these q_{ij} are constant, two interpretations can be given to the rate of movement from state i to state j. These are shown in Figure 2.1. In the interpretation on the left, the sojourn time in state i is distributed as $q_i = \mathsf{min}\{\mathsf{Exp}(q_{ij})\}$—since these are exponential distributions, this sojourn time is distributed as $F_{\mathsf{min}\{\mathsf{Exp}(q_{ij})\}}(t) = 1 - e^{-\sum_k q_{ik} t}$. Then the probability of changing to state j is given by $p_{ij} = q_{ij} / \sum_k q_{ik}$. The product of these two terms is exactly the conditional sojourn time in state i given that the next state is j.

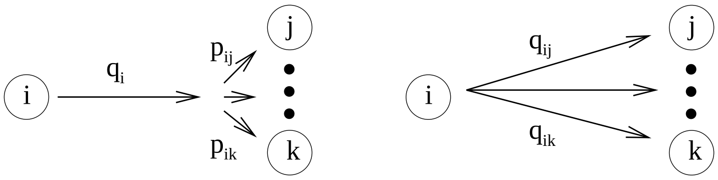

Figure 2.1: Two views of a state change in a CTMC

An aggregation of a Markov process is achieved by inducing a partition over the state-space of the process. Each partition of states from the original process provides a state in the aggregate process. The infinitesimal generator matrix of the lumped process can be formed by summing over the transition rates of states in a partition. However, in general, the steady-state solution of the aggregate process will be unrelated to the steady-state solution of the original. If the aggregate process is a *strictly lumpable partitioning* of the original process, then the equilibrium solutions are related—the probability of being in a state of the aggregate, Π_S, is given by the sum of the probabilities of being in the original states which make up the class, $\sum_{i \in S} \pi_i$. Two Markov processes are *lumpably equivalent* if they have lumpable partitions with equal numbers of elements, and there is a one-to-one correspondence between the partitions that matches the aggregate rates between the partitions (the degenerate case of one partition containing all states is disallowed). Lumpability is closely related to a process algebraic equivalence relation, and this is detailed in Section 2.3.5.

A more general stochastic process is the *semi-Markov process*. Formally, this can be viewed as a renewal process which, on each renewal, chooses a successor state based upon a discrete time Markov chain. This means that sojourn times in states may be arbitrarily distributed, and the process only possesses the Markov property on state changes. A semi-Markov process can be thought of as a generalisation of two simpler stochastic processes—if the state-space is of size 1, then it is a simple renewal process; if the state sojourn times are exponentially distributed, then it is a continuous time Markov chain. The semi-Markov process is a first step on the path to incorporating generally distributed random variables into stochastic process algebras. For more details on the nature of these processes, see [34, 52].

2.2.4.1 Balance Principles

The focus in this thesis is on stochastic processes in *steady-state*. This is characterised as a particular long-term behaviour of a stochastic process. A process in steady-state has the following properties. First, it is *irreducible*, that is, intuitively, that all states can be reached from all other states. Second, it is *ergodic*, meaning also that each of its states is positive-recurrent. Then it can be shown that there is a unique probability distribution, $\underline{\pi} = (\pi_1, \pi_2, \dots)$ such that

$$\underline{\pi} Q = \underline{0} \tag{2.2.9}$$

Intuitively, this means that the process reaches a situation where the probabilities of being found in any particular state remain fixed after some time t. If the process is in state k at time t, the distribution a short time later is given by

$$\pi_k(t + \Delta t) = \pi_k(t) + \big(\sum_{i=1}^{n} q_{ik}\pi_i(t)\big)\Delta t + o(\Delta t) \tag{2.2.10}$$

In the limit as $\Delta t \to 0$, this gives

$$d\underline{\pi}(t)/dt = \underline{\pi}(t)Q \tag{2.2.11}$$

At steady-state, $d\underline{\pi}(t)/dt = 0$, giving Equation (2.2.9) above. Let $\underline{\pi}$ represent $\underline{\pi}(t)$ at steady-state i.e. $\lim_{t\to\infty} \pi(t)$. Then the equation can also be written in the following form:

$$\pi_i \sum_{j\neq i} q_{ij} = \sum_{j\neq i} \pi_j q_{ji} \tag{2.2.12}$$

These equations are called the *global balance equations* (or *balance conditions*) of the Markov process in equilibrium, or at steady-state. Therefore at steady-state, π_i is the proportion of time that the process spends in state i. The *flux out* of a state is given by the probability of being in that state multiplied by the rate at which the state is left. The *flux in to* the current state is the sum over for each predecessor state, the probability of being in that state multiplied by the rate at which that state is left to arrive in the current state. Recall that residence time in a state of a continuous time Markov chain is exponentially distributed, and therefore q_{ij} is constant in t for all i, j.

It is possible to place more restrictive balance conditions on a Markov process, and this leads to some interesting properties. Consider the following equations, for all i, j:

$$\pi_i q_{ij} = \pi_j q_{ji} \tag{2.2.13}$$

If a Markov process satisfies this property, then it is said to be in *local balance*. This implies that when the time parameter of the process is *reversed*, the resulting process is statistically identical to the first process. The resulting stationary distribution of such a process is given by a simple product of ratios of instantaneous transition rates. Notice that any solution to these local balance equations is also a solution to the global balance equations. A slightly less strict set of local balance equations results in a property called *quasi-reversibility*. This property was first described in a queueing theory setting, for networks of queues, or queues with multiple classes of customers. Here, given any state i, a set of states $S(r,i)$ is specified, which represents those states with one more class r customer than i. The local balance equations are then:

$$\pi_i \sum_{j \in S(r,i)} q_{ij} = \sum_{j \in S(r,i)} \pi_j q_{ji} \tag{2.2.14}$$

For a multi-class open queueing network where each node is quasi-reversible, it can be shown that the equilibrium solution is a *product form* over the nodes of the queue. This is a powerful property, ensuring that the nodes are independent of each other, despite the traffic flow between them. Notice again that a solution to the local balance equations is also a solution to the global balance equations of the process.

More forms of local balance equation are employed in this thesis, in order to ensure *insensitivity* of a stochastic process. A stochastic process is said to be *insensitive* if its steady-state distribution depends on the distribution of one or more of its lifetime variables only through the mean. Insensitivity implies that all generally distributed lifetimes may be replaced by arbitrarily distributed lifetimes, and so long as the mean is preserved, the steady-state solutions of both processes will be identical. Typically, exponentially distributed lifetimes are used as replacements, due to their elegant mathematical properties. This then allows a conventional Markovian analysis to be carried out. Matthes [58] devised a new stochastic model, the *Generalised Semi Markov Scheme*, a generalisation of a semi-Markov process. The process resides in a state, in which multiple lifetimes are 'alive', and the 'death' of any one of these causes a change of state. Matthes showed that this model was insensitive to the distributions of these lifetimes if a particular set of balance equations was satisfied assuming all lifetimes are exponentially distributed (retaining their means). This result was proved in an alternative fashion by Schassberger [67], by using probabilistic *mixtures* of Erlang-k distributions to approximate generally distributed random variables. Continuity arguments were then used to extend this result to arbitrarily distributed random variables. An example of an insensitive process is the semi-Markov process. Intuitively, this process has only one lifetime per state; when this lifetime dies, the process changes state according to a discrete time Markov chain. The form of the distribution of this lifetime is unimportant, since the mean of the lifetime governs the steady-state probability of being present in that state. This observation is borne out by considering the set of insensitivity balance equations for any semi-Markov process—they are always consistent with (in fact equal to) global balance for the exponential version of the semi-Markov process. Chapters 4 and 5 make use of the insensitivity results above as a way to incorporate generally distributed random variables into PEPA.

2.3 Stochastic Process Algebras

In recent years, interest has grown in the use of a process algebra based methodology for performance modelling and evaluation. Stochastic process agebras (SPA) resulted from the development of classical process algebras by the inclusion of timing and probability. Systems are modelled by terms of the algebra, as an interaction of *agents*, or *processes* (not to be confused with stochastic processes, described earlier). The behaviour of each process is defined by the *activities* it can perform, or compositionally by the behaviour of its *subcomponents*.

In this section, the key features of SPA are described, focusing on PEPA, a prominent SPA, and the particular algebra used for the work in this thesis. The transition graph semantics of PEPA are described, which lead directly to an interpretation as a stochastic process. Finally, PEPA's major equivalence relations are introduced and defined.

2.3.1 Classical Process Algebras

Process algebra theory introduces processes as terms of an algebraic language which comprises a small number of basic combinators. These are used to compositionally describe the behaviour of a system. Two famous examples of classical process algebras are Milner's CCS [59] and Hoare's CSP [47]. Transitions such as $P \xrightarrow{\alpha} Q$, stating that process P may become Q by performing the *action* α underpin the behavioural meaning of a CCS or CSP process. Transitions are derived by structured rules, in that the behaviour of a compound process is inferred by operational rules from the behaviour of its subcomponents.

Processes are defined by terms of the algebra, and are built from broadly the same set of combinators (allowing for some differences in interpretation). For example, if α is an *action*, and P is a process, then by application of the CCS *action prefix* combinator, $\alpha.P$ is a process. Intuitively, this process is capable of performing an α action and evolving into P. P is called a (α-)*derivative* of $\alpha.P$. This intuition is formalised in the action prefix rule of CCS:

$$\frac{}{\alpha.E \xrightarrow{\alpha} E}$$

This has no premises (conditions above the line), and therefore is an axiom. E is a process variable, and so it states that any prefix process can make a transition signifying that an action has been performed. Another combinator of CCS is *choice*, e.g. $P + Q$ is a process which may perform an action enabled by either P or Q—the choice is competitive, and the derivative of $P + Q$ is either the derivative of P or Q, depending on which action is chosen. An important combinator for compositional modelling is the *parallel combinator*, $P \mid Q$, expressing that P and Q may proceed independently, or possibly cooperate on particular activities by handshaking. Each of these combinators has a corresponding rule, defining precisely which transitions may be inferred. PEPA has an analogue of each of these combinators, and its particular interpretation will be given later.

The semantics of a CCS process is given operationally, in the style of Plotkin [61], and results in a *labelled transition system*. The semantics is interleaving, in that the independent parallel composition of two processes is treated as a choice between the possible sequential orderings of the enabled actions. CCS models are *qualitative* in that they express no timing information, only the relative ordering of actions. They may also be said to be *reactive*, in that the possible behaviours of a CCS process may be constrained by placing it in an environment, another process, with which it must cooperate.

2.3.2 Adding Random Variables

Classical process algebras do not provide facilities that allow the modelling of a system as a stochastic process. This motivated the creation of stochastic process algebras, which feature one critical difference—a duration is associated with each action. A timed action is represented by two items of data—an *action type*, such as α, and an *activity rate*, such as r; together an activity $\mathsf{a} = (\alpha, r)$. The value r represents the parameter of an exponential random variable, and therefore each activity is deemed to have an exponentially distributed 'time-to-fire', or 'time-to-completion'. The distribution function of a is denoted F_{a} and is given by $\mathsf{Exp}(r)$. If a process enables several activities $(\alpha_1, r_1), (\alpha_2, r_2), \ldots$, the behaviour is governed by a 'race-condition'; one activity will complete first, and the time until completion of the race will be distributed as the minimum of the rates of the enabled activities, i.e. $\mathsf{min}\{\mathsf{Exp}(r_i)\}$. This *execution policy* means that the non-determinism present in CCS is replaced by probabilistic branching in PEPA; the probability that (α_i, r_i) completes is given by $\Pr(\mathsf{Exp}(r_i) < \mathsf{min}\{\mathsf{Exp}(r_j) : j \neq i\}) = r_i / \sum_{j \neq i} r_j$. Such a process can be said to be *generative* in that it is completely specified and can evolve independently of an environment [69].

A process algebra which features timed activities governed only by exponential random variables is also termed a *Markovian process algebra*. In this thesis, a and b will typically range over activities, α and β over action types, and r and s over activity rates. Activity rates range over $\mathbb{R}^+ \cup \{r\top : r \in \mathbb{R}^+\}$, where $\mathbb{R}^+$ is the set of positive real numbers. The symbol $\top$ means that a rate is *passive*, a concept explained in the next section. The set of all action types is denoted $\mathcal{A}$, and the set of all activities is denoted by $\mathcal{A}ct$.

Three Markovian process algebras are prominent in the literature. MTIPP [30] was developed at the University of Erlangen, and EMPA [8] was developed at the University of Bologna. Both of these algebras have been developed to include *immediate actions*, that is, actions that are resolved without the passing of time. This idea is not discussed further in this thesis. PEPA [44], developed by Hillston at the University of Edinburgh, has a concise theory, and will be used to illustrate the concepts behind SPA.

2.3.3 Syntax and Semantics of PEPA

As is the case for CCS, the behaviour of a PEPA process, and how it interacts with the environment, is determined by a small set of combinators. While most Markovian process algebras agree on the kind of combinators that should be provided, they disagree on some particular details.

Definition 2.3.1 (Syntax of PEPA). *Let A be a set of* process constants*, and let $A_S \subseteq A$ be a set of* sequential process constants*. The syntax of PEPA processes is given by*

$$\begin{aligned} S &::= (\alpha, r).S \mid S + T \mid A_S \\ P &::= P \underset{L}{\bowtie} Q \mid P/L \mid A \mid S \end{aligned} \tag{2.3.1}$$

This is a two-level grammar. Any process described by S is termed a *sequential component*. A process P consists of a *model configuration* of sequential components. The combinators described by P persist over process transitions; the combinators described by S do not. This distinction between levels is vital when considering a PEPA model as a stochastic process in steady-state. Due to this property, a PEPA model may be described as a sequence of sequential components, so long as a record is kept of the static structure. For example, if P is the process $((\alpha, r).R \underset{L}{\bowtie} (S/L)) \mid\mid T$, then it may be denoted $\langle(\alpha, r).R, S, T\rangle_P$. Each PEPA combinator is described briefly below:

Prefix: if P is a process, then $(\alpha, r).P$ is a process that performs (α, r) (an activity of type α, exponentially distributed with mean $\frac{1}{r}$) and then evolves into P.

Summation/Choice: if P and Q are processes, then $P + Q$ is a process that expresses the conflicting competition of P and Q. The current activities of both P and Q are enabled; a race condition determines the first to complete and distinguishes the component into which the process evolves.

Constant: if P is a process and $A \stackrel{\text{def}}{=} P$, then A is a process that behaves in exactly the same fashion as P.

Cooperation: if P and Q are processes, and L is a set of action types, then $P \underset{L}{\bowtie} Q$ is a process that expresses the parallel and synchronising execution of both P and Q. Both components proceed independently on activities whose types are not in the cooperation set, L. However, those activities whose types are contained in L require the participation of both P and Q. If one of the components may not perform the activity in its current state, the other component becomes *blocked* on that activity. If both are capable of performing the activity, the activity may occur with a rate which reflects the rate of the slower participant.

If the cooperation set L is empty, the parallel composition of P and Q is denoted $P \mid\mid Q$. More generally, the parallel composition of a set of I-indexed processes is given as $\prod_{i \in I} P_i$.

Hiding: if P is a process, and L is a set of action types, then P/L is the process that can behave exactly as P, except that if P would perform an activity of type $\alpha \in L$, then P/L would perform a *silent* activity denoted by the type τ. Activities whose types are in L are said to be *hidden*, and cooperation is not possible on τ activities.

A classical process algebra combinator missing from PEPA is the nullary combinator $\mathbf{0}$. This is the *deadlocked* process, which is incapable of performing any action. This is a deliberate omission from PEPA, because such a process can only assist in writing process descriptions which may lead to absorbing states in the underlying performance model. To date, the focus with PEPA modelling has been on steady-state 'long-run' performance measures, and $\mathbf{0}$ has no useful place in such processes. Note that it is still possible to provide a description of a PEPA process which is incapable of performing any activity—an example is given in Equation (2.3.4).

Within any given PEPA model, several activities with the same action type, say α, may be enabled, and will race to complete, but to an external observer, there will be a single rate at which activities of type α complete. This is called the *apparent rate* of α. The following definition is from [44].

Definition 2.3.2 (Apparent Rate). *The* apparent rate *of action type α in component P is denoted $r_\alpha(P)$, and is given by:*

$$\begin{aligned}
r_\alpha((\beta, r).P) &= \begin{cases} r & \textit{if } \alpha = \beta \\ 0 & \textit{otherwise} \end{cases} \\
r_\alpha(P + Q) &= r_\alpha(P) + r_\alpha(Q) \\
r_\alpha(P/L) &= \begin{cases} r_\alpha(P) & \textit{if } \alpha \notin L \\ 0 & \textit{otherwise} \end{cases} \\
r_\alpha(P \underset{L}{\bowtie} Q) &= \begin{cases} r_\alpha(P) + r_\alpha(Q) & \textit{if } \alpha \notin L \\ \min(r_\alpha(P), r_\alpha(Q)) & \textit{otherwise} \end{cases}
\end{aligned}$$

This means that the apparent rate of an activity of type α enabled on both sides of a choice is given by the sum of the rates of the activities; they compete in a race. However, for a cooperation, where α is in the cooperation set, the apparent (witnessed) rate of α is given by the minimum of the rates possible by each component. Therefore the slowest participant determines the rate of the cooperation.

A PEPA activity can also be given a *passive* rate, denoted by $\top$. An activity of the form $\mathsf{a} = (\alpha, \top)$ does not determine the rate at which a occurs—in fact, a must occur in cooperation with another enabled a, the cooperating component determining the rate at which the activity occurs. If the semantics of PEPA determine that a particular process P enables a passive activity, then P is called *incomplete*, and on its own cannot provide a basis for a performance evaluation. It is possible to give a simple syntactic algorithm which will always detect if a PEPA process is incomplete; however it may also reject as incomplete a PEPA process which is not. Currently, there is no known syntactic procedure which will always determine if a PEPA process is not incomplete. Passive rates may appear in apparent rate expressions, and so the following inequalities and equations specify how such expressions may be manipulated:

$$\begin{aligned}
r &< w\top && \text{for all } r \in \mathbb{R}^+ \text{ and for all } w \in \mathbb{N} \\
w_1\top &< w_2\top && \text{if } w_1 < w_2, \text{ for all } w_1, w_2 \in \mathbb{N} \\
w_1\top + w_2\top &= (w_1 + w_2)\top && \text{for all } w_1, w_2 \in \mathbb{N} \\
\frac{w_1\top}{w_2\top} &= \frac{w_1}{w_2} && \text{for all } w_1, w_2 \in \mathbb{N}
\end{aligned}$$

Figure 2.2 presents the operational semantic rules for PEPA. Collectively, the set of rules is called **PEPA Rules**. The formal semantics of a PEPA process is given by a *labelled multi-transition system* $(\mathcal{C}, \mathcal{A}ct, \{\xrightarrow{(\alpha,r)} : (\alpha, r) \in \mathcal{A}ct\})$, where $\mathcal{C}$ is the set of all PEPA components, and the relation $\xrightarrow{(\alpha,r)}$ is the least relation that can be inferred using **PEPA Rules**.

Prefix
$$\frac{}{(\alpha, r).E \xrightarrow{(\alpha,r)} E}$$

Choice
$$\frac{E \xrightarrow{(\alpha,r)} E'}{E + F \xrightarrow{(\alpha,r)} E'} \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E + F \xrightarrow{(\alpha,r)} F'}$$

Coop
$$\frac{E \xrightarrow{(\alpha,r)} E'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E' \underset{L}{\bowtie} F} \,(\alpha \notin L) \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E \underset{L}{\bowtie} F'} \,(\alpha \notin L)$$

$$\frac{E \xrightarrow{(\alpha,r_1)} E' \quad F \xrightarrow{(\alpha,r_2)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,R)} E' \underset{L}{\bowtie} F'} \,(\alpha \in L)$$

where $R = (r_1/r_\alpha(E))(r_2/r_\alpha(F)) \min(r_\alpha(E), r_\alpha(F))$

Hide
$$\frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\alpha,r)} E'/L} \,(\alpha \notin L) \qquad \frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\tau,r)} E'/L} \,(\alpha \in L)$$

Const
$$\frac{E \xrightarrow{(\alpha,r)} E'}{A \xrightarrow{(\alpha,r)} E'} \,(A \stackrel{\text{def}}{=} E)$$

Figure 2.2: Operational semantic rules of PEPA

The semantics of PEPA ensure that the transition systems for T and R below are equivalent:

$$\begin{aligned} P &\stackrel{\text{def}}{=} (\alpha, r).P & \qquad R &\stackrel{\text{def}}{=} (\alpha, r).R_1 + (\beta, s).R_2 \\ Q &\stackrel{\text{def}}{=} (\beta, s).Q & R_1 &\stackrel{\text{def}}{=} (\beta, s).R \\ T &\stackrel{\text{def}}{=} P \mid\mid Q & R_2 &\stackrel{\text{def}}{=} (\alpha, r).R \end{aligned} \tag{2.3.2}$$

This is an *interleaving* semantics. This may seem at odds with the notion of timed activities, since the intuition for T is that the enabled activities do not interfere with each other, and when one completes, the other continues without being affected. However, R insists that when one completes, the other is then *restarted*. These views are consistent with each other due to the fact that activities are distributed exponentially. Therefore, when one activity completes, the time for which another has been enabled can be statistically dismissed—the distribution is memoryless. This means that for PEPA, a *resume* semantics is equivalent to a *restart* semantics. Non-interleaving semantics have been developed for stochastic process algebras. For example, Katoen [51] develops a quantitative extension of *event structures*, which provides a partial order of process terms expressing only the necessary causality between activities, and not arbitrary interleavings. These semantics have the advantage that models grow linearly as the number of cooperating components increase—a transition graph semantics grows exponentially, motivating work on the *state-space explosion* problem. However, systematically mapping an event structure model to a stochastic process remains as work to be addressed.

Now some basic definitions regarding transitions are given. If $P \xrightarrow{(\alpha,r)} P'$ then P' is called a ((α, r)-) (one-step) *derivative* of P. If $P \xrightarrow{(\alpha,r)}$, then there exists some P' such that P' is a (α, r)-one-step derivative of P. If $P \longrightarrow$ then there exists some (α, r) such that $P \xrightarrow{(\alpha,r)}$; if $P \longrightarrow P'$ then there exists some (α, r) such that $P \xrightarrow{(\alpha,r)} P'$. If $P \xrightarrow{\alpha} P'$ then there is some rate r such that $P \xrightarrow{(\alpha,r)} P'$. Finally, ${}^{\diamond}P$ is used to range over processes P' such that $P' \longrightarrow P$.

Definition 2.3.3 (Derivative set). *The* derivative set *of a PEPA process P is denoted $ds(P)$, and is the smallest set of components such that*

- $P \in ds(P)$
- *if $P' \in ds(P)$ and $P' \longrightarrow P''$, then $P'' \in ds(P)$.*

The derivative set of a process captures the set of all reachable derivatives. The derivative set is used in the definition of the *derivation graph* of a PEPA process.

Definition 2.3.4 (Derivation graph). *The* derivation graph *of a PEPA process P is a labelled directed multi-graph (G, E), where $G = ds(P)$, and E is a multiset of labelled arcs such that $(P', P'', (\alpha, r)) \in E$ with the same multiplicity as the number of distinct inference trees which imply $P' \xrightarrow{(\alpha,r)} P''$.*

The (multi-)set of all activities possible by a process P is denoted $\overrightarrow{\mathcal{A}ct}(P)$, and is equal to $\biguplus_{P' \in ds(P)} \biguplus_{\{\mathtt{a} \,:\, P' \xrightarrow{\mathtt{a}}\}} \mathtt{a}$.

2.3.4 Deriving a Continuous Time Markov Chain

The derivation graph of a PEPA model may be used to give the model a stochastic process semantics, enabling performance evaluation to be carried out. A PEPA model can be interpreted as a continuous time Markov chain as described below.

Theorem 2.3.1 (PEPA Model as a CTMC [44]). *For any finite PEPA model P, define the stochastic process $\{X_t : t \in T\}$ such that $X_t = P' \in ds(P)$ implies that the process behaves as P' at time t. Then $\{X_t : t \in T\}$ is a continuous time Markov chain.*

Consider any derivative, $P' \in ds(P)$. P' will enable some number of activities, $\mathtt{a_i} = (\alpha_i, r_i)$, each of which is exponentially distributed. The syntactic terms are the states of the stochastic process, and the conditional sojourn time in a state, $S_{P',\mathtt{a}}(t)$, is the probability the time spent as P' will be at most t given that the sojourn ends with the completion of activity $\mathtt{a}$. Recall the duration of each activity $\mathtt{a_i}$ is exponentially distributed as $F_{\mathtt{a_i}}(t) = 1 - e^{-r_i t}$. It is then straightforward to show that the unconditional sojourn time in any state, $S_{P'}(t)$ represents an exponentially distributed random variable with parameter $\sum_k r_k$, and therefore this process has the Markov property.

Now some notation is presented. Given a derivative P, the *exit rate* (or *departure rate*) from P is the parameter of the distribution governing the sojourn time in P. It is denoted $q(P)$. The *transition rate* between two components P and P' is given by

$$q(P, P') = \sum_{\{\!|(\alpha,r) \,:\, P \xrightarrow{(\alpha,r)} P'|\!\}} r \tag{2.3.3}$$

These $q(P, P')$ provide the off-diagonal elements of the infinitesimal generator matrix of the continous-time Markov chain, Q.

As stated, in this thesis performance analysis is meant with respect to models in steady-state only. Some conditions must be placed on the semantics of a PEPA model to ensure it results in a stochastic process with an equilibrium solution.

Definition 2.3.5 (Cyclic PEPA process). *A PEPA process P is* cyclic *if for all $P' \in ds(P)$, $P \in ds(P')$.*

This implies that behaviour may always be repeated, and therefore that the stochastic process underlying P is irreducible. If the semantics are finite, then the process is also guaranteed to be ergodic, and thus the stochastic process is guaranteed to have a unique equilibrium probability distribution over the derivatives of the process.

A PEPA model must be cyclic in order to be ergodic, but this condition is not sufficient for it to be ergodic. To see this, consider the following PEPA process definitions:

$$\begin{aligned} P &\stackrel{\text{def}}{=} (\alpha, r).P' \\ Q &\stackrel{\text{def}}{=} (\beta, s).Q' \\ R &\stackrel{\text{def}}{=} P \underset{\{\alpha,\beta\}}{\bowtie} Q \end{aligned} \tag{2.3.4}$$

This process is *deadlocked*, since it cannot proceed with any activity. Such PEPA processes are illegal when performing a steady-state analysis.

2.3.5 Equivalence Relations

A key feature of a process algebra is a notion of equivalence between two processes. First of all, this requires a commitment to the nature of equivalence over processes. Several notions have been proposed. The simplest of these is called *trace semantics* [47]. The idea is that two processes are considered equivalent if they can perform identical 'execution-traces', where a trace is simply a finite sequence $\langle \mathtt{a}_1, \mathtt{a}_2, \ldots \rangle$ of activities. If a two processes have identical (possibly infinite) trace sets, then they are considered equal. This notion is unsatisfactory because a process $\mathtt{a}.(\mathtt{b}.P + \mathtt{c}.Q)$ has the option of performing either an $\mathtt{b}$ or $\mathtt{c}$ activity of type after it has performed $\mathtt{a}$. However, the trace equivalent $\mathtt{a}.\mathtt{b}.P + \mathtt{a}.\mathtt{c}.Q$ commits to either $\mathtt{b}$ or $\mathtt{c}$ by performing $\mathtt{a}$. Therefore, when a process is considered to be observable, that is able to interact with an environment, the former process is less restricted than the latter. This led to the development of equivalence relations based on *bisimulation*, which were adapted for use with Markovian process algebras.

Bisimulation aims to capture the notion of the equivalence of two processes as determined by an observer. The following definition is a bisimulation relation for PEPA analagous to that for a classical process algebra.

Definition 2.3.6. *$\mathcal{R}$ is a* bisimulation *relation if for P, Q processes, and $(P, Q) \in \mathcal{R}$, then for all $\mathtt{a} = (\alpha, r) \in \mathcal{A}ct$,*

- *Whenever $P \xrightarrow{\mathtt{a}} P'$, then for some Q', $Q \xrightarrow{\mathtt{a}} Q'$, and $(P', Q') \in \mathcal{R}$;*
- *Whenever $Q \xrightarrow{\mathtt{a}} Q'$, then for some P', $P \xrightarrow{\mathtt{a}} P'$, and $(P', Q') \in \mathcal{R}$.*

Notice that *at each stage*, if a process enables an activity, then its counterpart must be able to match it, and vice versa. Two processes P and Q are bisimilar if there is some bisimulation $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$.

While this candidate relation captures the qualitative observable behaviour of a PEPA process, it fails to be a suitable notion of equivalence because it equates two processes with wildly different *timing* behaviour. This may be mitigated by the addition of a condition to the definition:

$$r_\alpha(P) = r_\alpha(Q) \tag{2.3.5}$$

The addition of this extra condition results in PEPA's *strong bisimulation* relation. Two processes are *strongly bisimilar*, written $P \sim Q$, if there is some strong bisimulation relation which equates them. However, this relation still falls short. Consider the following PEPA process definitions:

$$\begin{aligned} P &\stackrel{\text{def}}{=} (\alpha, r).P + (\alpha, r).P + (\alpha, r).P' \qquad & Q &\stackrel{\text{def}}{=} (\alpha, r).Q + (\alpha, r).Q' + (\alpha, r).Q' \\ P' &\stackrel{\text{def}}{=} (\beta, s).P & Q' &\stackrel{\text{def}}{=} (\beta, s).Q \end{aligned} \tag{2.3.6}$$

P and Q satisfy the conditions required for being bisimilar. However, any observer also witnessing the passing of time would detect that $q(P, P') \times 2 = q(Q, Q')$, therefore, that the transition rates from two bisimilar processes to two bisimilar processes are different.

The solution for PEPA is an equivalence relation called *strong equivalence*. This is based on the *probabilistic bisimulation* of Larsen and Skou [56], which relates two processes if the *probability* of moving to a derivative in a set of probabilistically bisimilar processes is equal for the two processes. In PEPA, two processes P and Q are analogously *strongly equivalent* if some relation $\mathcal{R}$ forms equivalence classes of processes such that P and Q are both in the same class, and the rates at which P and Q both perform an activity of given type α to make a transition to any equivalence class are the same. Formally,

Definition 2.3.7 (Strong equivalence). *An equivalence relation $\mathcal{R} \subseteq \mathcal{C} \times \mathcal{C}$ is a* strong equivalence *if whenever $(P, Q) \in \mathcal{R}$, then for all $\alpha \in \mathcal{A}$, and $S \in \mathcal{C}/\mathcal{R}$,*

$$\sum_{P' \in S} q(P, P', \alpha) = \sum_{Q' \in S} q(P, Q', \alpha) \tag{2.3.7}$$

P and Q are said to be *strongly equivalent* if there is some strong equivalence relation $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$, and it is written $P \cong Q$. In particular, there is no strong equivalence relation relating the processes in Equation (2.3.6).

Strong equivalence has two crucial properties. The first is that it is a *congruence*; this means that it is preserved by all PEPA's combinators. Concretely, if $P \cong Q$, then P and Q can be placed in any PEPA 'context', intuitively a process algebra structure with a hole for a process, and the resulting two processes will be strongly equivalent. For example, $P \cong Q$ implies that $P \underset{L}{\bowtie} R \cong Q \underset{L}{\bowtie} R$ for any set of action types L, and process R. The reader is referred to [44] for a proof. This means that a modeller may take a PEPA model, and arbitrarily replace subcomponents with others that are strongly equivalent, and the result is a process that is strongly equivalent to the original. That this may be an advantage to the performance modeller is due to the second crucial property of strong equivalence—that strongly equivalent processes generate lumpably equivalent continuous time Markov chains. Recall that a lumped Markov process corresponds to an exact aggregation of a generally larger Markov process. In terms of PEPA processes, a process P which generates a stochastic process which is equal to the lumped process underlying Q will have a state-space no larger than Q. Therefore, this property provides the PEPA modeller with an exact aggregation technique—components in models can be replaced with strongly equivalent partners, resulting in a model with a smaller state-space, but with a lumpably equivalent and thus exactly aggregated stochastic process. In Chapter 3, contexts are formally introduced, and this property of strong equivalence is exploited to provide the performance modeller with an expressive method to specify performance measures which are preserved under strong equivalence. Each Markovian process algebra has a similar notion of strong equivalence—see [30, 8] for more details.

Other useful equivalence relations exist for PEPA. For example, *isomorphism* is a structural property, and two processes are *isomorphic*, written $P = Q$ if they generate equivalent derivation graphs. At the simplest level, *syntactic equivalence* between P and Q specifies that P is *exactly* Q, that is that P is a variable representing a process identical in structure to Q. This is denoted $P \equiv Q$, and is used often in this thesis to simplify presentation.

2.4 Reward Structures for Performance Measures

The performance of Markovian stochastic processes can be studied using a mathematical formalism called the *Markov reward model*. This model consists of two structures; a continuous time Markov chain, and a *reward structure*. The reward structure is introduced by Howard [49], and provides a very general framework for calculating performance measures. Its use is to specify rewards which accumulate as the Markov process evolves. Howard defines reward structures with respect to semi-Markov processes; in this full generality, a reward structure is:

- 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, $b_{ij}(\upsilon)$; when the transition from state i to state j is made at some time υ, the process earns the fixed reward $b_{ij}(\upsilon)$.

In most applications, the generality that this model affords is not needed, and some simplifications are made. In fact, for the applications described in this thesis, the yield rate is fixed to be constant during the occupancy of a state, and does not at any point depend on the choice of successor state. This gives

$$y_{ij}(\sigma) = \rho(i)$$

Moreover, bonus functions are discarded; therefore, a reward structure will consist of a constant yield function only. A simple method can be used in order to calculate some steady-state performance measures. Given a reward structure $\rho(i)$ associating a reward with state i, and π_i, a steady-state probability distribution as would be obtained by solution of a PEPA process, then

$$\sum_i \rho(i).\pi_i \tag{2.4.1}$$

gives a simple scalar value. This *reward* has different meanings, depending on the interpretation of the value of $\rho(i)$.

Alternative modelling paradigms make use of reward structures. For instance, stochastic Petri net models may have a reward structure specified over the reachability graph (which can be generated given a particular initial net marking). The use of reward structures is illustrated with the following simple example. Consider the SPN presented in Figure 2.3. Place P_3 acts as a semaphore, and ensures that places P_2 and P_5 cannot both contain tokens at the same time. In this way, the Petri net can be seen to model two processors, where each wishes to gain exclusive access to 'common memory'. The reachability graph is simple in this case, and is shown in Figure 2.4. An example of an interesting performance measure may be the throughput of accesses to common memory. In this case, the reward structure is defined such that a reward is assigned to each state that enables transition T_2 or T_4; the value assigned is the rate at which the state in the reachability graph is left. The dot-product construction above would then give the throughput value as required.

An exposition of the use of Markov reward models and reward structures is given by Haverkort and Trivedi [36]. The reward structure can be instantiated in different ways

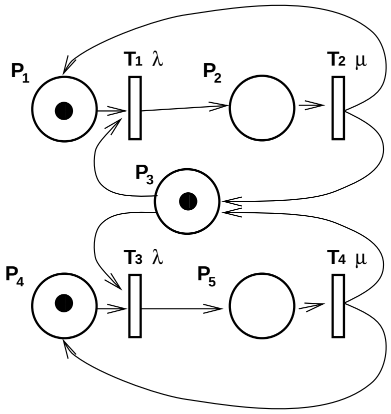

Figure 2.3: A simple stochastic Petri net example

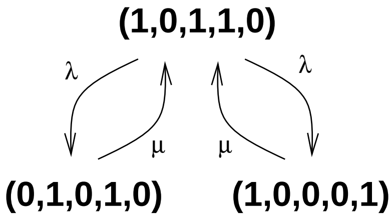

Figure 2.4: The reachability graph

so as to specify a wide range of performance variables. Let $\{X_t : t \geq 0\}$ be a finite-state CTMC, representing the evolution of a system in time. The reward rate of the system at time t is denoted by the random variable R_t. One of the simplest useful properties is the probability of completing a given amount of work in a specified time interval. This can be specified by the random variable Y, parameterised by t, meaning the accumulated reward until time t:

$$Y(t) = \int_0^t R_\tau d\tau$$

The questions asked of these models may be about cumulative or instantaneous behaviour. An example of a question about cumulative performance is simply 'what is the total output of the machine at time t?', and therefore would involve $Y(t)$, the accumulated reward. An alternative question is 'what is the rate of work now?', which would involve R_t, the instantaneous reward rate. However, the modeller may instead wish to address the long run behaviour of the system, for example 'what is the average throughput of the machine?' This would involve the *time-averaged* accumulated reward as time t tends to infinity, that is $\lim_{t\to\infty} E[Y(t)/t]$. The different families of performance models for which reward structures are employed are illustrated in Table 2.4, which is reproduced in part from [70]. *Reliability* models consider only two levels of performance—operational and failed. Therefore the range of

Model Family	Absorbing states	Reward Structure
Reliability	all have 0 reward	$\{0,1\}$
Availability		$\{0,1\}$
Imperfect repair	at least one has 0 reward	$\mathbb{R}^+ \cup \{0\}$
Guaranteed completion	none have 0 reward	$\mathbb{R}^+ \cup \{0\}$

Figure 2.5: Families of Markov reward models

the reward structure is simply $\{0,1\}$. Reliability models are such that all and only states with 0 reward are absorbing. Haverkort and Trivedi define 'system reliability' as $Rel(t) = \Pr(R_\tau = 1, \forall \tau \in (0,t))$, that is the probability the system is not in a failure state in the interval $(0,t)$. Moreover, the long run availability may be denoted $Rel(\infty)$, that is the probability of finding the model in a non-failure state as t tends to infinity. *Availability* models differ in that 0 reward states need not be absorbing, and therefore they can capture the repair of failed components. A further generalisation is the class of *imperfect repair* models. The range of the reward structure for such models is the set of positive real numbers, $\mathbb{R}^+ \cup \{0\}$. Typically a non-empty set of states with 0 reward are absorbing, but the more general reward structure is intended to indicate a varying level of performance of the system, such as the failure of some, but not all, components. Finally, the set of models with the *guaranteed completion* property are all those such that no set of absorbing states has a 0 reward. This means that the probability of residing in a 0 reward state indefinitely is zero, and therefore any finite amount of reward will be accumulated if the time interval is long enough. For instance, models of fault-tolerant systems in which all system failure states can lead to a repair state are said to have the guaranteed completion property.

With this taxonomy, it can be readily seen that the Petri net example presented in Figure 2.3 is an example of a time-averaged accumulated reward. The model should have an irreducible subset of states if long run behaviour is to be studied, and in fact is ergodic. The reward structure takes values in the set $\mathbb{R}^+ \cup \{0\}$; states which enable T_2 or T_4 are assigned a reward equal to the rate at which these transitions are taken, and all other states receive the value 0.

When modelling using PEPA, the aim is to generate a model such that the underlying CTMC is ergodic. Since, then, every state is reachable from every other in finite time, it would be inappropriate to attempt to study models with imperfect repair. Indeed the focus is on the computation of steady-state measures, that is looking at the system long after all short-term effects have disappeared. In the literature, several systems have been described using PEPA, and reward structures have been constructed manually (by examining regular patterns in the state space of process expressions) and then used to calculate pertinent measures, such as throughput and utilisation, for example [48, 27, 11, 45].

Sanders and Meyer [66] present a wide range of example performance variables resulting in several classes of performance measures. Their modelling paradigm is the *stochastic automata network*, a model which incorporates features of both stochastic Petri nets and queueing models. By use of a more general reward structure, incorporating both yield and bonus functions, the authors demonstrate reliability, availability and throughput measures. Furthermore, their yield functions may be defined in terms of the current marking of the SAN—this provides them with an expressive way to isolate particular states which can be specified at the level of the SAN.

2.5 High-Level Models with Generally Distributed Random Variables

A theme of this thesis is the use of generally distributed random variables in PEPA, a Markovian process algebra. Much research is currently devoted to incorporating generally distributed random variables into other high-level modelling paradigms. An overview of the foundational work is presented here.

2.5.1 Stochastic Petri Nets

Several extensions of stochastic Petri nets have been proposed which introduce generally distributed random variables. The random variables are used to govern the firing time of net *transitions*. *Extended stochastic Petri nets* (ESPN) were proposed by Dugan *et al.* [26]. They allowed generally distributed transitions if the following rules were satisfied:

1. The firing time of non-conflicting transitions which are enabled concurrently must be exponentially distributed.

2. The firing time of an exclusive transition, a transition which is never enabled concurrently with another, may be generally distributed.

3. A transition in conflict may be generally distributed, but all others with which it conflicts must be exponentially distributed.

The intuition for these rules can be understood by examining the authors' choice of stochastic model, which they required to be a semi-Markov process (SMP). The first two conditions can be readily understood by recourse to the global balance equations of the stochastic process. The first rule ensures that at the level of the stochastic process, it is unnecessary to remember the spent lifetimes of transitions—general distributions are forbidden in this case. The first condition rules out generally distributed concurrently enabled non-conflicting transitions. The second rule allows a generally distributed transition when it is never enabled concurrently with another. The enabling of such a transition will be represented at only one state in the stochastic process; if the state change is the result of another transition firing, then the generally distributed transition must be cancelled by the rule, and will not need to resume in the successor state. The second condition listed above is a strong one; for such a transition T, every marking in which it is enabled represents a state in which no other transitions are enabled.

As described by Henderson and Lucic [37], the third condition can be understood as a practical concession which means that an SMP, which features general distributions, may still be generated and solved with little computational effort. The restriction leads to tractable next state probabilities, due to a result presented by Ajmone Marsan and Chiola [2]. In principle, all concurrently-enabled transitions which conflict may be generally distributed. When one fires, all others are disabled, and a realistic interpretation is that when re-enabled, each transition is assigned a new time-to-live. This means that for any such marking, a successor marking will not require a record of any transition's residual lifetime. Since the transitions notionally compete, the next marking should be chosen based on the transition which is fastest to fire. Therefore, the distribution of the sojourn time in a state of the SMP is given by the minimum of the distributions associated with each transition.

Deterministic and stochastic Petri nets (DSPN) were proposed by Ajmone Marsan and Chiola [2]. Such nets employ exponentially distributed and constant firing delays. A restriction is that in no marking may there be more than one deterministic transition enabled—this allows a tractable analysis of the DSPN. When a deterministic transition is enabled competitively with exponential distributions, the stochastic process underlying a DSPN is a semi-Markov process; when enabled concurrently with exponential distributions, the process is a *Markov regenerative stochastic process.* Such a process possesses the Markov property at *regeneration* points only; at the level of the DSPN, these points correspond to the firing of a deterministic transition. Ciardo *et al.* [14] and Lindemann [57] provide a comprehensive summary of this class of stochastic Petri nets and their underlying stochastic processes.

2.5.2 Stochastic Process Algebras

A popular approach to incorporating generally distributed random variables into SPA is to build a process algebra calculus where every activity may have a generally distributed lifetime. Such an approach has both benefits and disadvantages. The obvious and greatest advantage is the extra modelling flexibility this affords the user. There is no longer a requirement that model activities have durations which are exponentially distributed only.

One disadvantage is that some familiar process algebra rules are no longer applicable in general. For example, consider the parallel composition of two processes, each capable of performing a single activity. The familiar *expansion law* states, using informal notation, that $\mathtt{a} \mid\mid \mathtt{b}$ is *observationally* equivalent to $\mathtt{a}.\mathtt{b} + \mathtt{b}.\mathtt{a}$. However, when `a` and `b` are not modelled with memoryless distributions, the interleaving approach is incorrect. This is because after one activity completes, the choice does not represent the spent lifetime of the other activity.

When general distributions may be used arbitrarily in a process algebra model, it becomes very difficult, in general, to solve the process, if, for example, the user is interested in a steady-state probability distribution. Markovian models can be mapped to CTMCs, and these may be solved for steady-state using linear algebra. Non-Markovian process algebra models correspond to less restricted stochastic processes.

An early non-Markovian approach to process algebra was exemplified by Strulo and Harrison [35]. Their framework enhanced traditional process algebra with probabilistic and timed features, resulting in a model with several distinct transition relations. Their approach to performance evaluation was to show how their models could evolve over time via a discrete event simulation.

A recent example of a process algebra incorporating general probability distributions is *Generalised Semi-Markovian Process Algebra* (GSMPA), by Bravetti *et al.* [12]. Their calculus incorporates all the traditional process algebra combinators, including choice and parallel composition, and they provide a mapping to a GSMP, the stochastic process introduced in Section 4.2.1. GSMPA is provided with a ST semantics, meaning the evolution of an action is represented as a combination of action start and action termination. One ramification of this decision is that a choice among actions is governed by a *preselection* policy, essentially meaning the choice is over transitions representing action starts. This action partitioning is similar to that present in *Interactive Markov Chains* (IMC), presented in the thesis of Hermanns [41]. Hermanns separates actions into two disjoint sets—immediate actions, and exponentially distributed actions. His framework allows a theory of weak bisimulation which is a congruence over the operators of the language of IMC; familiar CCS-style weak bisimulation is used over immediate actions, and MPA-style strong equivalence is used over exponentially distributed actions. Of course, the models of IMC processes are interpreted as continuous-time, and when all non-determinism is resolved, lead to CTMCs. The same weak bisimulation approach is used in later work by Bravetti *et al.* [13]. Here they restrict GSMPA such that it is only possible to synchronise on untimed actions. The resulting calculus is called *Interactive Generalized Semi-Markov Processes* (IGSMP). The restriction allows a weak bisimulation result similar to IMC where the stochastic process remains a GSMP. Furthermore, the authors list as further work extending their equivalence such that collections of timed silent activities can be aggregated in similar fashion to non-timed silent activities. For example, since their calculus incorporates general distributions, a sequence of τ-actions could be reduced to a single τ-action distributed as the convolution of the distributions of those in the sequence.

In [12], Bravetti *et al.* give an example of a simple queueing system modelled in GSMPA, where the queue has a deterministic service time. They determine that the resulting GSMP is insensitive when particular states of the GSMP model are amalgamated, and then derive a CTMC which they are able to solve conventionally for steady-state. This is in contrast to the new derived combinator for PEPA presented in Chapter 4—it does not allow the modeller the freedom to arbitrarily use generally distributed activities. However, it means that the modeller need not assume that any of a particular set of activities used in the model are exponentially distributed. Moreover, the insensitivity is guaranteed and does not need to be determined by the modeller on a model by model basis.

An alternative stochastic process algebra is *spades*($\spadesuit$), introduced by D'Argenio *et al.* [21]. Once more, the authors choose to separate the stochastic timed behaviour of the model from the actions it performs. Again this immediately gives a more visible correspondence with a GSMP. For example, if P is a spades process, then so is $\{\!|\, C \,|\!\}P$ where $\{\!|\, C \,|\!\}$ represents a set of *clocks.* Each clock has a distribution function, and thus corresponds to an active element of a GSMP. $\{\!|\, C \,|\!\}P$ represents a process where all clocks in $\{\!|\, C \,|\!\}$ are set according to their distribution functions, and begin counting down. The spades process $C \mapsto P$ represents the process that may become P if the clock C has reached zero. This represents a state change in a GSMP when an active element reaches the end of its lifetime. Of course their calculus allows processes to be expressed in which clock settings persist over transitions, and thus spent lifetimes are respected. Moreover due to their separation of action and duration, they recover a form of the expansion law. However for performance evaluation, they do not attempt to use analytical techniques, and instead choose discrete event simulation.

Finally, El-Rayes *et al.* [28] propose a distinctive approach to using general distributions in process algebra. They propose $\text{PEPA}^{\infty}_{\text{ph}}$, a modification of PEPA, such that activities can be distributed by phase-type probability distributions, and furthermore, consider models representing potentially infinitely many customers in a queueing system. The solution of an infinite-state model relies on it having a restricted structure—it must be decomposable into an initial portion, and a repetitive portion. The stochastic process underlying such a restricted $\text{PEPA}^{\infty}_{\text{ph}}$ model will then have a (infinite) generator matrix with a particular repeating structure, and can be solved by using matrix geometric methods. Despite features in $\text{PEPA}^{\infty}_{\text{ph}}$ which are superficially similar to those present in the work of this thesis, the methods presented are quite dissimilar. The models built in Chapter 4 implicitly contain models of queues, but none may contain an arbitrary number of customers. On the other hand, the theory of *insensitivity* employed need not restrict attention to phase-type distributions only. Furthermore, the models in this thesis can be solved in an entirely conventional way, as if all activities were exponentially distributed, a property guaranteed by the theory of insensitivity.

2.6 Tools for Performance Evaluation with SPA

In this section, the various tools available for performance modelling with SPA are described. The focus is on tools for modelling with PEPA, especially since the main tool is extended to incorporate some of the theory presented in Chapter 3.

2.6.1 The PEPA Workbench

The PEPA Workbench [29, 18] processes a textual description of a PEPA model, and generates output which can be used to calculate a steady-state probability distribution for the model. The Workbench performs some well-formedness checks on the model as it attempts to internally generate a derivation graph. These checks include ensuring that the model is built correctly according to PEPA's grammar, and that no deadlocks are possible (and therefore that the process underlying the model is ergodic).

Once the derivation graph is built, the Workbench generates a data file which represents the infinitesimal generator matrix of the PEPA model. In Figure 2.6, the PEPA model has two components which execute independently in parallel. Each component has three states and so the generator matrix for the corresponding Markov process has dimension nine. Even with an example as small as this one, it is apparent that a modeller would not wish to construct a Markov process matrix by hand for fear of introducing errors. The matrix does not represent the structure of the PEPA model, and contains rates only.

$$
\boxed{\begin{array}{lcl}
P_1 & \stackrel{\mathrm{def}}{=} & (\mathtt{start}, r_1).P_2 \\
P_2 & \stackrel{\mathrm{def}}{=} & (\mathtt{run}, r_2).P_3 \\
P_3 & \stackrel{\mathrm{def}}{=} & (\mathtt{stop}, r_3).P_1 \\
 & & \\
 & P_1 \parallel P_1 &
\end{array}}
\Rightarrow
\begin{pmatrix}
-2r_1 & r_1 & r_1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -r_1-r_2 & 0 & r_2 & r_1 & 0 & 0 & 0 & 0 \\
0 & 0 & -r_1-r_2 & 0 & r_1 & r_2 & 0 & 0 & 0 \\
r_3 & 0 & 0 & -r_1-r_3 & 0 & 0 & 0 & r_1 & 0 \\
0 & 0 & 0 & 0 & -2r_2 & 0 & r_2 & r_2 & 0 \\
r_3 & 0 & 0 & 0 & 0 & -r_1-r_3 & r_1 & 0 & 0 \\
0 & r_3 & 0 & 0 & 0 & 0 & -r_2-r_3 & 0 & r_2 \\
0 & 0 & r_3 & 0 & 0 & 0 & 0 & -r_2-r_3 & r_2 \\
0 & 0 & 0 & r_3 & 0 & r_3 & 0 & 0 & -2r_3
\end{pmatrix}
$$

Figure 2.6: The task of the PEPA Workbench

The modeller can make a choice of generating the matrix in a form suitable for processing by various mathematical software packages, for example, Maple or Mathematica, or in a more primitive form suitable for processing directly by a small program which implements the biconjugate gradient algorithm (see [62] for more details). These tools are all able generate a steady-state distribution $\underline{\pi}$, given the generator matrix Q, subject to the constraints

$$\underline{\pi}Q = \underline{0} \text{ and } \sum_i \pi_i = 1 \tag{2.6.1}$$

The Maple package has the added advantage that for reasonably small state-spaces, it is able to compute a steady-state probability distribution in terms of one or more symbolic rates. The textual description of the model above would be of the form illustrated in Figure 2.7.

```
# P_1 = (start,r_1).P_2 ;
# P_2 = (run,r_2).P_3 ;
# P_3 = (stop,r_3).P_1 ;

P_1 || P_1
```

Figure 2.7: Sample input to the PEPA Workbench

2.6.1.1 The PEPA State Finder

The PEPA State Finder is a tool to allow the modeller to explore the state space of a model, with a view to building a reward structure in order to generate performance measures. It provides a simple regular-expression pattern language for matching the syntax of a PEPA model description. For example, given the example presented in Figure 2.6, a pattern of the form '$P_1 \parallel *$' would match against $P_1 \parallel P_i$ for $i = 1, 2, 3$. In this way a particular subset of the state space may be isolated, simplifying the construction of a reward function. The method is straightforward to use, but suffers from some drawbacks—these are discussed in Chapter 3, where an alternative and more expressive technique is presented.

2.6.2 The TIPPtool

The TIPPtool is a modern graphical tool which supports the performance evaluation of Markovian TIPP models. The input language is LOTOS-based [9], and the tool provides facilities for *functional analysis* (for example, reachability analysis), and for the calculation of both steady-state and transient measures. It should be noted that the language of Markovian TIPP provides explicit support for deadlocked processes, with the goal of transient analysis in mind. The TIPPtool can thus calculate properties such as the mean time to absorption in a particular state (which is represented by a deadlock term). Built in to the TIPPtool are numerical routines able to solve the generator matrix underlying a TIPP model to generate a steady-state solution. The TIPPtool is capable of performing compositional minimisation of a TIPP model using any of several variants of a strong equivalence style relation. Finally, in order to generate steady-state performance measures, the TIPPtool also allows areas of the state-space to be isolated by use of regular expressions over the syntax of process terms.

2.6.3 TwoTowers

The Markovian process algebra EMPA has tool support for performance evaluation in the shape of the TwoTowers [7] application. TwoTowers makes use of two existing tools, the Concurrency Workbench of North Carolina [64], and MarCA [71]. The Concurrency Workbench is a tool designed for qualitative analysis with classical process algebra, providing facilities like model-checking, and reachability analysis. MarCA is a tool designed specifically for the stationary and transient analysis of Markov processes. TwoTowers simply provides an interface to each of these tools, thereby eliminating the requirement for producing custom written solvers and analysers. TwoTowers can process models representing EMPA_r processes—EMPA_r is an extension of EMPA to enable the expression of rewards in the process algebra syntax. This method of reward specification is discussed in Chapter 3.

Chapter 3

Performance Specification Techniques for SPA

In this chapter, a technique is presented which allows a class of performance measures to be automatically derived from SPA performance models. 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 often carried out in an ad hoc manner. Clearly this will inhibit the uptake of SPAs for performance modelling. Filling this gap with a rigorous method for specifying and calculating performance measures from SPA models would strengthen a weak stage in the SPA modelling methodology. The method proposed is for the specification and automatic calculation of steady-state performance measures only. To use the technique, the performance modeller provides a specification, the language of which is based upon a modified modal logic. The modeller can furthermore choose to study the behaviour of a set of subcomponents within the context of the model. This specifies a reward structure which may be automatically generated, and the resulting Markov reward model can be solved to derive the measure.

The chapter begins with a critique of current reward specification techniques for SPA.

3.1 Reward Specification with SPA

In this section, two previously published methods of reward specification, specifically for SPA, are discussed, with a breakdown of their strengths and weaknesses. The problems each has suggest features that an alternative specification method might possess. Such an alternative, the PEPA Reward language, is described in Section 3.2.

3.1.1 Using Regular Expressions

In [42], Hermanns and Mertsiotakis describe the reward specification mechanism used in their TIPPtool, an application for analysing SPA performance models. Given an SPA model, the CTMC is formed by a standard construction from the labelled transition system, itself formed using operational inference rules. Then in order to build a reward structure, they use a regular expression over the syntax of the process algebra term. In essence, this treats the algebra term like a string of characters. For example, consider the following SPA term:

$$P \underset{L}{\bowtie} Q \quad \text{where} \quad P \stackrel{\text{def}}{=} (\alpha, r).(\beta, s).P \tag{3.1.1}$$

The regular expression '$\_ * (\beta, *).P*$' would match all algebra terms that contained a process prefix term enabling an activity of type β, and which then evolves into P ('$\_$' is used to denote whitespace). The syntax of regular expressions used in the TIPPtool is similar to that provided by the Unix tool, `grep`, e.g. $*$ expresses '0 or more syntactic characters', and '$e_1 \mid e_2$' matches a string if either e_1 or e_2 matches. Once matched, a separate method is used to assign a value to these states in the reward structure. For example, the PEPA State Finder tool [18] provides a characterisation of the matching states in the form of a function, parameterised on the steady-state vector, which can be evaluated by a computer algebra package. The modeller may then manipulate the reward as appropriate. The regular expression technique is 'complete' in the sense that it can be used to select any combination of states in the transition system, and thus in the Markov chain. This is because it is trivial to write a regular expression that picks out syntactically one process algebra term only, and the regular expression language has a disjunction combinator. Of course this could become infeasible for any reasonably sized model; the idea of the technique is that one regular expression may capture a number of 'interesting' process algebra terms (and thus Markov chain states), and with these states a common value can be associated. Moreover, most users are familiar with the language of regular expressions, and this makes the method simple to pick up and use.

However, although this method is convenient, and has proven useful, it certainly is not ideal for a number of reasons. Most of the criticisms below stem from identifying states of interest by studying the syntax of a process algebra model, rather than by using the semantics. Firstly, it does not seem clear how a user would capture all terms enabling an activity of type α. If the ability of the model to perform an α indicated the availability of a particular resource, this would be a reasonable requirement. Again, consider the model presented in Equation (3.1.1). A regular expression such as '$*(\alpha, *).*$' would suffice to pick out the derivative $(\alpha, r).(\beta, s).P \underset{L}{\bowtie} Q$; however, this may equivalently be written as $P \underset{L}{\bowtie} Q$, which exhibits no syntactic occurrence of α. Another problem with the use of regular expressions is that they disregard the structural aspects of process algebra that allow, for instance, commutativity of terms in a synchronisation. For example, behaviourally, the term $Q \underset{L}{\bowtie} P$ is equivalent to Equation (3.1.1); many process algebra equivalences exist, but no practical equivalence distinguishes these two terms, since this is counter-intuitive. Unfortunately, it may be the case that distinct regular expressions are required to capture such permutations of process ordering. This suggests that a suitable reward specification technique for process algebra will focus on the visible *behaviour* of components, rather than their syntactic structure.

The lack of theoretical fit between process algebra and regular expressions is highlighted further by consideration of algebraic manipulations. Equivalence relations may be used to show that one process is behaviourally and performance equivalent to an *aggregated* process. The transition system of the aggregated process will typically be smaller than that of the original, since conceptually, each state of the aggregate consists of a set of states of the original. The strong equivalence relation ensures that these states are grouped together in such a way that the behaviour of the process is equivalent to the other in any context, and that the rate of departure from an aggregate state to another, is commensurate with the performance of the original. A PEPA modeller would rightly expect to be able to replace a component of a model with another strongly equivalent component, without altering the model's stochastic behaviour. However, if the modeller chooses to use regular expressions to specify the reward structure, it may be the case that the same regular expression will not do the same job in both cases. Consider the two processes in Equation (3.1.2) to be strongly equivalent.

$$\begin{aligned} P &\stackrel{\text{def}}{=} (\alpha, 2r).P' \\ Q &\stackrel{\text{def}}{=} (\alpha, r).Q' + (\alpha, r).Q' \end{aligned} \tag{3.1.2}$$

An observer will not be able to differentiate two larger PEPA processes, where one has P placed in a context, and the other has Q. Any regular expression of the form '$(\alpha, 2r).*$' will select P, but not Q; any weakened expression of the form '$(\alpha, *).*$' runs the risk of selecting other components enabling an activity of type α. The solution here is to have a specification mechanism which can examine the *timing capacity* of components to make particular transitions to equivalent components.

By the arguments above, it can be seen that by using regular expressions, it is easy to capture states which should not be included in calculations, and to miss out states which should be included. Certainly it seems that the match between regular expressions and process algebra models is uneasy.

3.1.2 Instrumenting SPA Models with Rewards

An alternative reward specification technique for SPA is advocated by Bernardo [5]. The method uses the Markovian process algebra EMPA; however variants of this method have been described for both PEPA, in [44], and TIPP, in [42]. The idea is not to use a separate formalism for specifying rewards, but rather to extend the language of the process algebra itself. This is done by adding a parameter to activities, denoting a reward value; for instance, (α, r) becomes (α, r, ρ), meaning that any model state enabling this activity has a contribution of ρ in the reward structure. It is assumed that rewards are *additive*; therefore, if a model state represents a process algebra term which enables two activities, (α, r, ρ) and (β, s, κ), the reward assigned to that state will be $\rho + \kappa$.

In this chapter, discussions of this technique will make use of PEPA. Consider the simple model below:

$$CPU \stackrel{\mathrm{def}}{=} (\mathtt{work}, w, w).(\mathtt{idle}, i, \mathit{0}).CPU$$

Suppose the throughput of this model was required. The *Forced Flow Law* states that the throughputs in all parts of a model must be proportional to each other. It can be seen that the throughput of the model is determined by the throughput of either activity; therefore, a reward of w is associated with the `work` activity (the activity was chosen arbitrarily; a reward of i could have been associated with `idle`). With r denoting the reward structure, and assuming a simple enumeration of the states of the transition system, the total reward would be calculated as $\sum_i r_i \cdot \pi_i$. Since $r_i = w$ in those states enabling `work`, this correctly gives the throughput as the rate at which useful work is carried out when possible, multiplied by the probability of being in a state in which useful work can be done.

Bernardo's extended process algebra is called EMPA$_r$. In [5], the traditional notion of MPA equivalence is extended to incorporate the reward structure induced by EMPA$_r$. The new relation is called *strong extended Markovian reward bisimulation equivalence*, denoted $\sim_{EMRB}$; this is $\sim_{EMB}$, EMPA's equivalent of PEPA's strong equivalence, with an extra feature that behaves as follows. $\mathcal{R}$ is a *strong extended Markovian reward bisimulation* iff $P\mathcal{R}Q$ implies that P and Q are strongly equivalent, and for each action type `a` the rewards P and Q accrue on making an `a`-transition to some equivalence class are the same. $P \sim_{EMRB} Q$ if P and Q are related by some strong extended Markovian reward bisimulation equivalence. Bernardo shows that $\sim_{EMRB} \subset \sim_{EMB}$, meaning the new equivalence relation is finer-grained; but the relation is also claimed to be a congruence, meaning equivalent terms, which accrue the same reward, also accrue the same reward in any process algebra context. Subsequent to the publication of this result, D'Argenio and Hermanns demonstrated that the congruence property does not actually hold for the full class of EMPA terms, in particular in cases where equivalent processes which enable different numbers of passive activities of the same action type are then forced to synchronise with an exponentially distributed activity. In his thesis [6], Bernardo establishes that despite this problem, the congruence result holds for a large and useful class of EMPA, and by extension EMPA$_r$, terms. Due to Bernardo's approach, it is straightforward for a modeller to incorporate the specification of rewards into EMPA models, and this is a definite strength of EMPA$_r$.

However, it can be argued that attaching rewards to activities both lacks expressive power, and can become cumbersome to use. This can be shown with a simple example; consider the PEPA model below:

$$\begin{aligned} CPU_i &\stackrel{\text{def}}{=} (\mathtt{get_i}, \lambda_i).(\mathtt{release}, \mu_i).CPU_i \\ SHMemory &\stackrel{\text{def}}{=} (\mathtt{get_1}, \top).(\mathtt{release}, \top).(\mathtt{get_2}, \top).(\mathtt{release}, \top).SHMemory \\ &(CPU_1 \parallel CPU_2) \underset{\{\mathtt{get_i},\mathtt{release}\}}{\bowtie} SHMemory \end{aligned} \tag{3.1.3}$$

The shared memory resource will be accessed by two processes, CPU_1 and CPU_2, performing the activities $\mathtt{get_1}$ and $\mathtt{get_2}$; however these processes must take turns. Suppose the modeller wished to find the utilisation of the resource by CPU_1. The shared memory is held by CPU_1 when the first instance of `release` is enabled, but not when the second is enabled. The modeller would thus have to realise that only the first instance of the `release` activity should be instrumented with a reward value of 1; effectively, the modeller would have to study the behaviour of *SHMemory* which is non-local to the instances of `release`. Better would be a specification method capable of assigning rewards based on more complex process behaviour. Next consider a modeller attempting to calculate the throughput of accesses to shared memory, given *SHMemory*. Unfortunately, although *SHMemory* defines the *behaviour* of the shared memory, its activities are passive and therefore the processors decide for how long they require access. Therefore the modeller has to study the processors *together* with the shared memory, in order to decide which CPU activities represent accesses, and instrument these activities only with rates for rewards (λ_1, and λ_2). If either processor uses $\mathtt{get_i}$ for a purpose other than accessing shared memory, this problem becomes non-trivial for the modeller.

In a comparison between PEPA and generalised stochastic Petri nets [25], it is suggested that the reward specification for PEPA requires improvement, and highlights the usefulness of basing a reward on states which enable two different activities. Both techniques above struggle with this simple requirement. The proposal developed in this chapter is based upon the ability to be this flexible, by exploiting a logic for studying the dynamic behaviour of models.

3.2 Results for Constructing the PEPA Reward Language

In this section, some background results are presented which underpin the use of the PEPA Reward language. The idea behind the PEPA Reward language is to use a modal logic, capable of expressing properties of a process, in order to determine a specific reward structure over a Markovian model. This idea was initially discussed by Hermanns in an extended abstract [40], but other than in published work presented in this thesis [15, 19] does not seem to have been developed elsewhere. There is much active research on the topic of logics for the verification of probabilistic systems. Hansson and Jonsson [32] present a logic called PCTL, based upon the qualitative temporal logic CTL [20]. PCTL is capable of expressing properties such as 'after `a` happens, with 95% probability `b` happens within t time units'. This means a property can be proven true over a chosen fraction of the execution paths of the model. However, this logic is interpreted over discrete time Markov chains. Later work by Baier and Kwiatkowska [3] introduces non-determinism meaning that an execution of the model, a computation tree, is determined with respect to a *scheduler*, which resolves non-determinism. De Alfaro and Manna [23] adapt discrete time logic verification rules to a continuous setting. Their models are *timed transition systems*, the temporal behaviour of which can be represented by discrete traces, or continuous traces. A discrete trace is a snapshot of the state of the system at a sequence of time points, and a continuous trace gives the state of the system over a sequence of intervals of the real line. They demonstrate that if a temporal logic formula is *finitely variable*, its validity over discrete traces implies its validity over corresponding continuous traces. However, transitions in their models are specified by minimum and maximum delays, which is unsuitable for working with continuous time Markov chains, where exponential random variables have no maximum firing time. Aziz *et al.* [1] present an alternative logic called CSL, specifically for verifying properties of continous time Markov chains. Again similar to CTL, it can express properties such as the PCTL example given above. Their semantics leads to definite integral terms over exponentials. By using number theoretic arguments, their result is that model checking CSL is effective, but they only speculate on the practicality of such a procedure.

In contrast to these methods, the logic chosen in this chapter is not temporal. A temporal logic features operators for, for example, expressing long-run properties, or for expressing properties that hold indefinitely often into the future. The aims of this work differ, in that the goal is not verification as such, but rather to provide an expressive language for specifying steady-state performance measures. PEPA models may be large, but are only relevant for steady-state analysis when finite; in practise, the expression of arbitrary temporal properties is more than is required here.

A modal logic formula will give a behavioural specification of the performance property of interest. In the same way that a classical temporal logic formula expresses a predicate over states of a transition system, a simple modal logic formula itself will express a predicate over the states of the transition system underlying the PEPA model. A specific set of states will be capable of the behaviour specified by the logical formula, and the others will not. Of course, this suggests that in order to calculate the partitioning, a model checking procedure should be employed. In the next section, a particular modal logic is studied, and it is shown why it is an appropriate choice for use with PEPA. Following this, the rôle of the logic in the Reward language is defined. The PEPA Reward language allows the modeller to exploit the compositionality present in a PEPA model, and to focus attention on particular subcomponents of a model. Some results are presented which highlight the extent to which this may be used in partnership with model aggregation.

3.2.1 A Logical Foundation for the Reward Language

The PEPA Reward language was first presented in [15]. The basis of the technique is a specification logic, used to describe the behaviour of process algebra models. The particular logic chosen then was Hennessy-Milner logic (HML) [39], a simple modal logic suitable for verifying non-temporal properties of *classical* models i.e. qualitative models without probabilities, random variables, or explicit reference to the passing of time. The syntax of HML formulas is given by

$$F ::= \mathtt{tt} \mid \neg F \mid F_1 \wedge F_2 \mid \langle\alpha\rangle F \tag{3.2.1}$$

The semantics below are the 'obvious' translation of the classical semantics given with respect to, for example, models of CCS processes. They indicate why this logic may not be ideal for working with PEPA processes. Formally, the semantics is given over the labelled multi-transition system of a PEPA model.

Definition 3.2.1. *Let P be a PEPA model. Then $\models_{\mathrm{HML}}$ is a* satisfaction relation *defined as the least relation satisfying the following rules:*

$$\begin{aligned}
&P \models_{\mathrm{HML}} \mathtt{tt} \\
&P \models_{\mathrm{HML}} \neg F \text{ if } P \not\models_{\mathrm{HML}} F \\
&P \models_{\mathrm{HML}} F_1 \wedge F_2 \text{ if } P \models_{\mathrm{HML}} F_1 \text{ and } P \models_{\mathrm{HML}} F_2 \\
&P \models_{\mathrm{HML}} \langle\alpha\rangle F \text{ if } P \xrightarrow{\alpha} P' \text{ and } P' \models_{\mathrm{HML}} F
\end{aligned} \tag{3.2.2}$$

The intuitive meaning of most of the modal operators is clear. $P \models_{\mathrm{HML}} \langle\alpha\rangle F$ if *there exists* a derivative P', such that P can make an α-transition to P', at any rate, and $P' \models_{\mathrm{HML}} F$.

In [39], Hennessy and Milner show that if two (classical) processes are strongly bisimilar, then they satisfy the same HML formulas. Furthermore, the converse holds, so long as the processes are image-finite. This result reinforces the prevalent view of bisimulation as a natural process equivalence, and also suggests that HML is very suitable as a classical process logic.

The most useful equivalence relation over PEPA models is strong equivalence. Recall that strongly equivalent processes generate lumpably equivalent Markov chains, and therefore the equivalence relation forms the basis of an exact model aggregation technique. Consider the following simple PEPA processes:

$$\begin{aligned} P &\stackrel{\text{def}}{=} (\alpha, r).(\beta, s).P \\ Q &\stackrel{\text{def}}{=} (\alpha, 2r).(\beta, s).Q \end{aligned} \tag{3.2.3}$$

These models are *not* strongly equivalent. Both generate Markov chains with two states, but for the first chain, the steady-state probability of being in the state corresponding to P is $s/(s+r)$, while in the second chain, the steady-state probability of being in the state corresponding to Q is $s/(s+2r)$. However, P and Q cannot be distinguished by any HML formula. The proof of this simple fact is omitted, but is clear because P and Q only differ syntactically in the rate of one activity and no HML formula refers to rates.

When the stated aim of the logic is to specify process behaviour, it seems that HML may not be an ideal choice. It is reasonable to expect that a PEPA modeller may wish to identify derivatives based on activity rates. These issues can be addressed by adapting a more appropriate logic, namely Larsen and Skou's PML [56].

Presented here is a modified version of the original Reward language, based on PML. The syntax of PML formulas is given by

$$F ::= \mathtt{tt} \mid \nabla_\alpha \mid \neg F \mid F_1 \wedge F_2 \mid \langle\alpha\rangle_p F \tag{3.2.4}$$

The models described in [56] are *probabilistic*, in that for any state P and any action α, there is a (discrete) probability distribution over the α-successors of P. Informally, the semantics of a formula ∇_α is the set of states unable to perform an α activity; and the semantics of $\langle\alpha\rangle_p F$ is the set of states such that each can make an α-transition with probability *at least* p to a set of successors each of which satisfies F. In this chapter, the interpretation of these formulas is modified slightly for use with PEPA models. First, two simple definitions:

Definition 3.2.2. $\mu_{P,\alpha}(Q) = \sum\{\!|r : P \xrightarrow{(\alpha,r)} Q|\!\}$

Definition 3.2.3. $P \overset{(\alpha,\nu)}{\Longrightarrow} S$ *if and only if* $P \xrightarrow{\alpha}$, *and* $\sum_{P' \in S} \mu_{P,\alpha}(P') = \nu$

In this thesis, a modified interpretation will be given to PML. It makes use of the above definitions so that the logic is suitable for use in a framework based on rates rather than probabilities. The modal logic which results will be called PML_μ. Now let P be a model of a PEPA process. The semantics of a PML_μ formula is a set of states for which the formula is satisfied. As is conventional, these semantics are again expressed by use of a satisfaction relation.

Definition 3.2.4. *Let P be a PEPA model. Then $\models_{\mathrm{PML}_\mu}$ is a* satisfaction relation *defined as the least relation satisfying the following rules:*

$$P \models_{\mathrm{PML}_\mu} \texttt{tt}$$
$$P \models_{\mathrm{PML}_\mu} \neg F \textit{ if } P \not\models_{\mathrm{PML}_\mu} F$$
$$P \models_{\mathrm{PML}_\mu} F_1 \wedge F_2 \textit{ if } P \models_{\mathrm{PML}_\mu} F_1 \textit{ and } P \models_{\mathrm{PML}_\mu} F_2$$
$$P \models_{\mathrm{PML}_\mu} \nabla_\alpha \textit{ if } P \overset{\alpha}{\not\longrightarrow}$$
$$P \models_{\mathrm{PML}_\mu} \langle\alpha\rangle_\mu F \textit{ if } P \overset{(\alpha,\nu)}{\Longrightarrow} S \textit{ for some } \nu \geq \mu, \textit{and for all } P' \in S, P' \models_{\mathrm{PML}_\mu} F \quad (3.2.5)$$

Henceforth, any use of $\models$ is assumed to refer to $\models_{\mathrm{PML}_\mu}$. The subscript μ present in formulas of the form $\langle\alpha\rangle_\mu F$ is interpreted as a rate rather than, as Larsen and Skou require, a probability. If a state P is capable of doing activity α *quickly enough* arriving at a set of states S each of which satisfies F, then P satisfies $\langle\alpha\rangle_\mu F$.

3.2.2 Relation of PML_μ to PEPA

By using PML_μ to underpin the PEPA Reward language, one of the main criticisms of the published work ([15]) will be addressed—that the (qualitative) logic employed was badly suited to the models. PML_μ formulas are able to distinguish model states that differ only in the rate at which they may perform activities. However it is also important to establish how this logic relates to PEPA. In [56], Larsen and Skou show that PML exactly characterises *probabilistic bisimulation*, in the sense that two probabilistic processes are bisimilar if and only if they satisfy exactly the same set of PML formulas. An analogous result holds for PEPA processes and PML_μ formulas:

Theorem 3.2.1 (Modal characterisation of strong equivalence). *Let P be an image-finite model of a PEPA process. Then*

$$P \cong Q \textit{ iff for all } F, P \models F \textit{ iff } Q \models F$$

Proof. The first case shows that $P \cong Q$ implies that for all F, $P \models F$ iff $Q \models F$; the second case shows the reverse of this implication.

Case 1 Assume $P \cong Q$. The proof proceeds by induction on the size of F, as in [56].

> **Case $F \equiv \langle\alpha\rangle_\mu G$:** Let $P \models F$. Then by definition, there exists a set S such that $P \overset{(a,\nu)}{\Longrightarrow} S$, where $\nu \geq \mu$, and for all $P' \in S$, $P' \models G$.
>
> Since $P \cong Q$, there exists some strong equivalence $\mathcal{R}$, such that for all $\alpha \in \mathcal{A}$, for all $S \in \mathcal{C}/\mathcal{R}, q[P,S,\alpha] = q[Q,S,\alpha]$. For each $P' \in S$, let $\mathcal{R}_{P'}$ be the equivalence class in $\mathcal{C}/\mathcal{R}$ which contains P'. Furthermore, let $S'' = \bigcup_{P' \in S} \mathcal{R}_{P'}$. Now, for each $P'' \in S''$, $P''\mathcal{R}P'$, and thus $P'' \cong P'$, for some $P' \in S$, and so by the hypothesis, for all $P'' \in S''$, $P'' \models G$.
>
> Since $S \subseteq S''$, $P \overset{(a,\nu')}{\Longrightarrow} S''$, where $\nu' \geq \nu$. Since $P\mathcal{R}Q$, for all $T \in \mathcal{C}/\mathcal{R}$, $q[P,T,\alpha] = q[Q,T,\alpha]$. However, note that for all $s, s' \in S$, $\mathcal{R}_s = \mathcal{R}_{s'}$ or $\mathcal{R}_s \cap \mathcal{R}_{s'} = \emptyset$. Therefore, by construction of S'', $q[P,S'',\alpha] = q[Q,S'',\alpha]$. Therefore, $Q \overset{(a,\nu')}{\Longrightarrow} S''$, where $\nu' \geq \nu \geq \mu$; and for all $Q' \in S''$, $Q' \models G$. Therefore, $Q \models \langle\alpha\rangle_\mu F$. By symmetry of $\cong$, this case is complete.
>
> **Case $F \equiv \nabla_\alpha$:** Let $P \models F$. Therefore $P \not\xrightarrow{\alpha}$. Since $P \cong Q$ it is the case that for some strong equivalence $\mathcal{R}$, for all $a \in \mathcal{A}$, for all $S \in \mathcal{C}/\mathcal{R}, q[P,S,\alpha] = q[Q,S,\alpha]$. However, for all $S' \subseteq 2^{\mathcal{C}}$, $q[P,S',\alpha] = 0$. Since for all $S \in \mathcal{C}/\mathcal{R}$, $S \subseteq 2^{\mathcal{C}}$, it is the case that $q[Q,S,\alpha] = q[P,S,\alpha] = 0$ for any $S \in \mathcal{C}/\mathcal{R}$, for any $\mathcal{R}$ which is a strong equivalence. Therefore there does not exist a C such that $q[Q,C,\alpha] > 0$ and therefore, $Q \not\xrightarrow{\alpha}$. By symmetry of $\cong$, this case is complete.
>
> All other cases are straightforward.

Case 2 Assume that for all F, $P \models F$ if and only if $Q \models F$.

> Let $\mathcal{R} = \{(P,Q)\colon$ for all F, $P \models F$ if and only if $Q \models F\}$. The result will hold if $\mathcal{R}$ can be shown to be a strong equivalence. By simple inspection, $\mathcal{R}$ is clearly an equivalence relation. Thus, it must be shown that for all $\alpha \in \mathcal{A}$, for all $S \in \mathcal{C}/\mathcal{R}$, $q[P,S,\alpha] = q[Q,S,\alpha]$.
>
> Let $S \in \mathcal{C}/\mathcal{R}$. Assume that for some $a \in \mathcal{A}$, $P \overset{(a,\mu)}{\Longrightarrow} S$ for some μ. Now consider the α-derivatives of Q, labelled $Q'_1, \ldots, Q'_m, Q'_{m+1}, \ldots, Q'_n$, where $n \geq 0$, $0 \leq m \leq n$. These derivatives are labelled such that $Q'_1, \ldots, Q'_m \in S$ and $Q'_{m+1}, \ldots, Q'_n \notin S$. For each Q'_i, let the rate at which Q makes an α-transition to Q'_i be denoted by μ_i. Since S is an equivalence class under $\mathcal{R}$, it is the case that for each $Q'_j, m+1 \leq j \leq n$, there exists a formula F'_j such that $Q'_j \models F'_j$, and for each $Q'_i, 0 \leq i \leq m$, $Q'_i \not\models F'_j$. From a lemma by Larsen and Skou [56], it is

possible to construct a dual formula to each F'_j, named here $\overline{F}'_j$ such that for $Q'_j, m+1 \leq j \leq n$, $Q'_j \models F'_j$ if and only if $Q'_j \not\models \overline{F}'_j$. Now it is the case that $Q \models \langle a \rangle_{\mu'}(\overline{F}'_{m+1} \wedge \ldots \wedge \overline{F}'_n)$, where $\mu' = \sum_{i=1}^{m} \mu_i$. However, by the initial assumption, it is also the case that $P \models \langle a \rangle_{\mu'}(\overline{F}'_{m+1} \wedge \ldots \wedge \overline{F}'_n)$, and therefore, $\mu' \geq \mu$. This then gives that $q[Q, S, \alpha] = \mu' \geq \mu$. However, $\mathcal{R}$ is symmetric, and so by such an argument, it is the case that $\mu \geq \mu'$, and thus that $\mu = \mu'$. Hence $q[P, S, \alpha] = q[Q, S, \alpha]$, as required.

□

The theorem shows that two PEPA processes are *strongly equivalent* (in particular, their underlying Markov chains are lumpably equivalent) if and only if they both satisfy the same set of PML_μ formulas.

Some PML_μ derived combinators are introduced in Equation (3.2.6). These add no expressive power to the logic, but will prove more succinct in expressing particular properties later.

$$\begin{aligned}
\mathtt{ff} &\stackrel{\text{def}}{=} \neg\mathtt{tt} \\
[\alpha]_\mu F &\stackrel{\text{def}}{=} \neg\langle\alpha\rangle_\mu\neg F \\
\Delta_\alpha &\stackrel{\text{def}}{=} \neg\nabla_\alpha \\
F_1 \vee F_2 &\stackrel{\text{def}}{=} \neg((\neg F_1) \wedge (\neg F_2)) \\
F_1 \implies F_2 &\stackrel{\text{def}}{=} (\neg F_1) \vee F_2 \\
F_1 \iff F_2 &\stackrel{\text{def}}{=} (F_1 \implies F_2) \wedge (F_2 \implies F_1)
\end{aligned} \tag{3.2.6}$$

From working with this logic, it appears that PML_μ does lack an HML combinator that is often useful in practice. PML_μ lacks the ability to express HML formulas of the form $\langle\alpha\rangle F$—as it stands, the diamond combinator requires decoration with a rate. Larsen and Skou [56] define the combinator intuitively as

$$\langle\alpha\rangle F \equiv \exists\mu > 0.\langle\alpha\rangle_\mu F \tag{3.2.7}$$

In analogous fashion to [56], for a PEPA model P the *minimal rate assumption* holds if there exists some ϵ such that for every $P' \in ds(P)$, there does not exist $\mu < \epsilon$ such that $P' \xrightarrow{(\alpha,\mu)}$. This is clearly satisfied for every PEPA model of interest in this thesis, since each is finite-state (and each is assumed to have a steady-state probability distribution). In light of this, the following abbreviations are also employed.

$$\begin{aligned}
\langle\alpha\rangle F &\stackrel{\text{def}}{=} \langle\alpha\rangle_\epsilon F \\
[\alpha] F &\stackrel{\text{def}}{=} \neg\langle\alpha\rangle\neg F
\end{aligned} \tag{3.2.8}$$

The PEPA Reward language, to be explained later in this chapter, will allow the modeller to make use of PML_μ formulas, leading to the automatic construction of a reward structure. Since the Reward language is used at the level of the process algebra, there are certain algebraic properties which should hold for the performance measures generated. As discussed in Chapter 2, strong equivalence is the most important of PEPA's equivalence relations; it is a congruence at the algebra level, and the application of strong equivalence aggregation can generate smaller, lumpably equivalent and exactly aggregated Markov chains. A crucial property for a PEPA modeller is that a performance measure generated for a particular model is identical to the measure generated for an aggregated model—Hillston [44] writes 'If the integrity of these measures is to be maintained by the strong equivalence relation, it must be possible to derive the same reward from the lumped derivation graph'. This property allows a separation of concerns; strong equivalence aggregation can be applied without knowledge of performance measures. Nicola [60] formalises this property by extending strong lumpability to Markov reward models:

Definition 3.2.5. *A Markov reward model is* strongly lumpable *with respect to a reward Θ in the context of a partition χ, if, for every starting distribution, the aggregated process is a Markov reward model which results in the same reward.*

Intuitively, a MRM is strongly lumpable with respect to a reward function if the underlying stochastic process of the MRM is strongly lumpable, and by using the reward function, the steady-state performance measures calculated for both processes are equal.

Let P, Q be PEPA models, and let $ds(P)/\cong$ give the state space of the lumped Markov chain of a model P. Let F be a PML_μ formula, and consider any function $R(\cdot)$ with the following property:

$$P \cong Q \implies R(P) = R(Q) \tag{3.2.9}$$

Now let $\rho(\cdot)$ be a reward function such that

$$\rho(P) = \begin{cases} R(P) & \text{if } P \models F \\ 0 & \text{otherwise} \end{cases} \tag{3.2.10}$$

The following proposition shows that any Markov process built from the derivation graph of a PEPA model is strongly lumpable with respect to such a reward function $\rho(\cdot)$.

Lemma 3.2.1. *The Markov reward model based on $ds(P)/\cong$ is strongly lumpable with respect to any reward function $\rho(\cdot)$ satisfying Equation (3.2.10) for $R(\cdot)$ satisfying Equation (3.2.9).*

Proof. The proof is straightforward, by the method of calculation of a steady-state measure given in Equation (2.4.1). The total reward is given by

$$\Theta = \sum_{P' \in ds(P)} \rho(P') \cdot \pi(P') = \sum_{S \in ds(P)/\cong} \Big(\sum_{P' \in S} \rho(P') \cdot \pi(P') \Big)$$

However, by Equation (3.2.9), for any PML_μ formula F, and any $P', P'' \in S$,

$$P' \cong P'' \implies P' \models F \text{ iff } P'' \models F$$

Therefore,

$$\begin{aligned} & \sum_{S \in ds(P)/\cong} \Big(\sum_{P' \in S} \rho(P') \cdot \pi(P') \Big) \\ &= \begin{cases} \sum_{S \in ds(P)/\cong} \Big(\sum_{P' \in S} R(P') \cdot \pi(P') \Big) = \sum_{S \in ds(P)/\cong} R \cdot \Pi_S & \text{if } P' \models F \\ \sum_{S \in ds(P)/\cong} \Big(\sum_{P' \in S} 0 \cdot \pi(P') \Big) = 0 & \text{otherwise} \end{cases} \end{aligned}$$

□

Proposition 3.2.1 illustrates that any such reward assignment function $R(\cdot)$ induces a reward-preserving equivalence relation that coincides with strong equivalence.

Corollary 1. *Let $\cong_r$ be an equivalence relation such that $P \cong_r Q$ if and only if function $R(\cdot)$ generates equivalent reward structures for P and Q. Then $\cong_r = \cong$.*

Proof. Immediate.

Therefore unlike the 'finer-grained' $\sim_{EMRB}$ relation for EMPA_r, any such reward preserving equivalence relation for PEPA will not discriminate between more processes.

3.2.3 Working with Subcomponents

The PEPA Reward language allows a modeller to work with subcomponents of a PEPA model, specifying a reward structure for the *whole* model dependent on behaviour *local* to a subcomponent working within the *context* of the rest of the model. In order to make this notion precise, a theory of *contexts* is employed. In this section, some definitions are borrowed from the work of Larsen [55, 54], who made use of contexts in work on context-dependent bisimulation. The following definitions and results are presented.

Definition 3.2.6 (Context). *Let A be a set of* process constants. *A PEPA* context *is a set of place-holders* (holes) [.] *in a PEPA process. The syntax of contexts is given below.*

$$c ::= [.] \mid A \mid (\alpha, r).c \mid c + c \mid c/L \mid c \underset{L}{\bowtie} c$$

The variables c, d and e will be used to range over contexts. This commonly used syntax is unfortunate for these purposes due to its clash with the syntax of PML_μ. A place-holder in a context, $[.]$, should not be confused with the derived logical box operator $[\alpha]$. An arbitrary context has a fixed number of holes, as determined formally by the following function.

Definition 3.2.7 (Number of holes). *Given a context c, $\mathsf{holes}(.)$ is a function giving the number of holes in c as*

$$\begin{aligned}
\mathsf{holes}([.]) &= 1 \\
\mathsf{holes}(P) &= 0 \\
\mathsf{holes}((\alpha, r).c) &= \mathsf{holes}(c) \\
\mathsf{holes}(c + c') &= \mathsf{holes}(c) + \mathsf{holes}(c') \\
\mathsf{holes}(c/L) &= \mathsf{holes}(c) \\
\mathsf{holes}(c \underset{L}{\bowtie} c') &= \mathsf{holes}(c) + \mathsf{holes}(c')
\end{aligned} \tag{3.2.11}$$

A method is required for placing processes (or more generally contexts) *within* a context; this is given by the following definition.

Definition 3.2.8 (Populating a context). $c[c_1, \ldots, c_n]$ *specifies a context with at most n holes populated, according to the following inductive definition:*

$$\begin{aligned}
([.])[c_1, \ldots, c_n] &= \begin{cases} c_1 & \text{if } n > 0 \\ [.] & \text{otherwise} \end{cases} \\
((\alpha, r).c)[c_1, \ldots, c_n] &= (\alpha, r).(c[c_1, \ldots, c_n]) \\
(c + c')[c_1, \ldots, c_n] &= \begin{cases} (c[c_1, \ldots, c_i]) + (c'[c_{i+1}, \ldots, c_n]), i = \mathsf{holes}(c) & \text{if } n > \mathsf{holes}(c) \\ (c[c_1, \ldots, c_n]) + c' & \text{otherwise} \end{cases} \\
(c/L)[c_1, \ldots, c_n] &= (c[c_1, \ldots, c_n])/L \\
(c \underset{L}{\bowtie} c')[c_1, \ldots, c_n] &= \begin{cases} (c[c_1, \ldots, c_i]) \underset{L}{\bowtie} (c'[c_{i+1}, \ldots, c_n]), i = \mathsf{holes}(c) & \text{if } n > \mathsf{holes}(c) \\ (c[c_1, \ldots, c_n]) \underset{L}{\bowtie} c' & \text{otherwise} \end{cases}
\end{aligned} \tag{3.2.12}$$

Example 1. *Let $c = (S + [.]) \underset{L}{\bowtie} ([.] \parallel [.])$. Then*

$$\begin{aligned}
c[T, P] &= ((S + [.])[T]) \underset{L}{\bowtie} ((([.] \parallel [.])[P]) \\
&= (S + ([.])[T]) \underset{L}{\bowtie} ((([.])[P] \parallel [.]) \\
&= (S + T) \underset{L}{\bowtie} ((([.])[P] \parallel [.]) \\
&= (S + T) \underset{L}{\bowtie} (P \parallel [.])
\end{aligned}$$

Of course, $c[[.]] = c$ for any context c. If a context has no holes, it is said to be *full*. If every subcontext of a context is not full, it is said to be *empty*. A context is called *dynamic* if it is an instance of $c + c'$ or $(\alpha, r).c$; it is called *static* if it is an instance of c/L or $c \underset{L}{\bowtie} c'$. A context c is called *non dynamic* (respectively *non static*) if every non-full subcontext of c is not dynamic (respectively not static). Notice that the empty context, [.], is neither non dynamic, nor non static.

Lemma 3.2.2. *Let c, c' be contexts with 1 and n holes respectively. Let c_i be a context, for $1 \leq i \leq n$. Then*

$$(c[c'])[c_1, \ldots, c_n] = c[c'[c_1, \ldots, c_n]]$$

Proof. Proof is by induction over the structure of c. Most cases are straightforward and omitted for brevity.

Case $c \equiv d + d'$: But

$$\begin{aligned}
(d + d')[c'][c_1, \ldots, c_n] &= (d[c'] + d')[c_1, \ldots, c_n] \text{ if } \mathsf{holes}(d) = 1 \text{ by Definition 3.2.8} \\
&= (d[c'][c_1, \ldots, c_n] + d') \text{ since } \mathsf{holes}(c') = n \\
&= (d[c'[c_1, \ldots, c_n]] + d') \text{ by inductive hypothesis} \\
&= c[c'[c_1, \ldots, c_n]] \qquad (3.2.13)
\end{aligned}$$

or alternatively

$$\begin{aligned}
(d + d')[c'][c_1, \ldots, c_n] &= (d + d'[c'])[c_1, \ldots, c_n] \text{ if } \mathsf{holes}(d) = 0 \text{ by Definition 3.2.8} \\
&= (d + d'[c'][c_1, \ldots, c_n]) \text{ since } \mathsf{holes}(c') = n, \mathsf{holes}(d) = 0 \\
&= (d + d'[c'[c_1, \ldots, c_n]]) \text{ by inductive hypothesis} \\
&= c[c'[c_1, \ldots, c_n]] \qquad (3.2.14)
\end{aligned}$$

□

With the simple result above, the following lemma illustrates when two occupied contexts are strongly equivalent.

Lemma 3.2.3. *Let c, c', c'' be contexts such that c has one hole, c'' has n holes, and $c[c''] = c'$. Let $P, P_1, \ldots, P_n$ be PEPA processes such that $P \cong c''[P_1, \ldots, P_n]$. Then*

$$c[P] \cong c'[P_1, \ldots, P_n]$$

Proof.

By assumption, $c'[P_1, \ldots, P_n] = c[c''][P_1, \ldots, P_n]$, and following by Lemma 3.2.2, $c[c''][P_1, \ldots, P_n] = c[c''[P_1, \ldots, P_n]]$. Now, strong equivalence guarantees that if $Q \cong R$, then $c[Q] \cong c[R]$ for all contexts c. Therefore, since $P \cong c''[P_1, \ldots, P_n]$, then $c[P] \cong c'[P_1, \ldots, P_n]$. □

The motivation for these results is to allow the PEPA modeller to implicitly work with processes *in context*. Of course a context is simply a PEPA process skeleton, and a fully specified model will always be represented by a full context. If c is a context, and $\underline{P} = P_1, \ldots, P_n$ are subcomponents such that $c[\underline{P}]$ is full, then the pair $(\underline{P}, c)$ is called a *view* of the process $c[\underline{P}]$, and each subcomponent P_i is *within the view* of c. Lemma 3.2.3 shows the relationship between two contexts c and c', one with one hole, and the other with n. It says that for any subcontext c'' of c' within which the n holes are located, if both c and c' are equivalent up to the subcontext c'', and it is the case that P is strongly equivalent to a fully populated c'', then $c[P]$ is strongly equivalent to a fully populated c'. When using the PEPA Reward language, the modeller is able to focus on a set of subcomponents of a larger model. This lemma will be used later to highlight which of these subcomponents can be replaced by strongly equivalent aggregated models, while guaranteeing the same reward structure with the same reward specification.

To generate a reward structure, the modeller begins by specifying a PML$_\mu$ formula. The modeller may also restrict attention to a specific set of subcomponents. Therefore, a meaning must be given to the satisfaction of a PML$_\mu$ formula over a set of subcomponents *in context*. From this point, attention is restricted to *non dynamic* contexts only. In a real sense, any process within a dynamic context does not represent a single subcomponent. Consider the process $P + Q$. Both P and Q are within a dynamic context, but when this process makes a transition, one side of the partnership will notionally have lost a race, and is eliminated. Therefore, components within a dynamic context do not generally persist over the evolution of a model (of course the resulting derivative may be built using a *new* choice context). Moreover, the dynamic context does not persist either, in general. In a similar fashion, a prefix context is guaranteed to vanish from a derivative, if the transition was due to the activity specified by the prefix.

The following lemma shows that if c is a non dynamic context with n holes, then when populated, any derivative P' can also be viewed as a populated non dynamic context c' with n holes.

Lemma 3.2.4. *Let c be non dynamic with n holes. Then if $c[P_1, \ldots, P_n] \xrightarrow{(\alpha,r)} P$, it is possible to express P as $c'[P'_1, \ldots, P'_n]$, for some non dynamic c', and in particular, c' has n holes. Moreover, for $1 \leq i \leq n$, and some $\beta \in \mathcal{A}$, either*

- *P_i did not contribute in the inference tree of $c[P_1, \ldots, P_n] \xrightarrow{(\alpha,r)} P$; or*
- *$P_i \xrightarrow{(\beta,s)} P'_i$ was used in the inference tree of $c[P_1, \ldots, P_n] \xrightarrow{(\alpha,r)} P$, for some s.*

Proof. The proof is by induction over the structure of c.

Case $c = [.]$**:** Then $n = 1$, and if $c[P_1] \xrightarrow{(\alpha,r)} P$, then $P_1 \xrightarrow{(\alpha,r)} P$. Therefore, $c' = [.]$ since $P \equiv c'[P]$. P_1 did contribute in the inference tree, and $\beta = \alpha$.

Case $c = c' \underset{L}{\bowtie} c''$**:** Without loss of generality, let c' have i holes ($0 \leq i \leq n$); then c'' has $n - i$ holes. Notice that both c' and c'' must be non dynamic. Since $c[P_1, \ldots, P_n] \xrightarrow{(\alpha,r)} P$, then by **PEPA Rules**, either

- $c'[P_1, \ldots, P_i] \xrightarrow{(\alpha,r)} P'$ for some P', and $P' \underset{L}{\bowtie} c''[P_{i+1}, \ldots, P_n] \equiv P$; or
- $c''[P_{i+1}, \ldots, P_n] \xrightarrow{(\alpha,r)} P''$ for some P'', and $c'[P_1, \ldots, i] \underset{L}{\bowtie} P'' \equiv P$; or
- $c'[P_1, \ldots, P_i] \xrightarrow{(\alpha,s)} P'$ for some P' and s, $c''[P_{i+1}, \ldots, P_n] \xrightarrow{(\alpha,t)} P''$ for some P'' and t, and $P' \underset{L}{\bowtie} P'' \equiv P$.

The third case is the most interesting, and the others follow by similar arguments. By the inductive hypothesis,

- it is possible to express P' as $c'''[P'_1, \ldots, P'_i]$, where c''' is non dynamic, and for $1 \leq j \leq i$, and some $\kappa \in \mathcal{A}$, either P_j was not used in the inference tree, or $P_j \xrightarrow{(\kappa,u)} P'_j$ for some u; and
- it is possible to express P'' as $c''''[P'_{i+1}, \ldots, P'_n]$, where c'''' is non dynamic, and for $i + 1 \leq j \leq n$, and some $\gamma \in \mathcal{A}$, either P_j was not used in the inference tree, or $P_j \xrightarrow{(\gamma,v)} P'_j$ for some v.

This leads to the following:

$$\begin{aligned} P &\equiv c'''[P'_1, \ldots, P'_i] \underset{L}{\bowtie} c''''[P'_{i+1}, \ldots, P'_n] \\ &\equiv (c''' \underset{L}{\bowtie} c'''')[P'_1, \ldots, P'_i, P_{i+1}, \ldots, P_n] \quad \text{by Lemma 3.2.2} \end{aligned} \tag{3.2.15}$$

It must now be shown that $\kappa = \gamma$. But since this case is due to a cooperation, $\alpha \neq \tau$. Therefore by the rules of PEPA, $\kappa = \alpha = \gamma$. Both c''' and c'''' are non dynamic by assumption, and therefore so is $c''' \underset{L}{\bowtie} c''''$. $c''' \underset{L}{\bowtie} c''''$ has $i + n - i = n$ holes, and the n arguments are as demonstrated above. If P_i was employed in the inference tree for a subcontext, then by the PEPA rule for cooperation, it is also employed in the inference tree for the cooperation (and similarly for the case when P_i is not used).

Other cases: Omitted.

□

One significant consequence of this lemma is that given a process $P \equiv c[P_1, \dots, P_n]$, every derivative $P' \in ds(P)$ can be expressed as some context c', with n holes, populated in order with for $1 \leq i \leq n$, some derivative $P_i' \in ds(P_i)$.

The next task is to capture the notion of subcomponents proceeding in context. The presentation follows Larsen [55], and gives an operational semantics to PEPA contexts. A context is viewed as an *action-transducers*, that is it is viewed semantically as an object which consumes activities produced by its internal processes and in return produces activities for an external observer. In this way, it acts as an *interface* between the two. Larsen uses the notation $c \xrightarrow[(\mathtt{a}_1 \dots \mathtt{a}_n)]{\mathtt{a}} c'$ to represent that the context c consumes the inner actions $\mathtt{a}_1$ to $\mathtt{a}_n$, produces the outer action $\mathtt{a}$, and changes into the new context c'.

Consider the cooperation operator of PEPA. The operational semantics of this operator can be presented contextually, showing how this operator performs as an activity transducer. From Figure 2.2, transitions may be inferred, for two processes combined with this operator, in three ways—either side may make a transition individually, or both may cooperate. Following Larsen, the cooperation may be interpreted in the following way: 'whenever the inner processes P_1 and P_2 produce activities (α, r_1) and (α, r_2), the cooperation combinator may combine these to produce the activity (α, R), where $R = (r_1/r_\alpha(P_1))(r_2/r_\alpha(P_2)) \min(r_\alpha(P_1), r_\alpha(P_2))$.' Since the combinator is static, it persists in the process which results from the transition. This means the behaviour of the combinator (indexed by a set of action types L) can be represented by the following *transduction*:

$$\underset{L}{\bowtie} \xrightarrow[\langle(\alpha, r_1), (\alpha, r_2)\rangle]{(\alpha, R)} \underset{L}{\bowtie} \tag{3.2.16}$$

Figure 3.1 gives a transduction semantics to all of the combinators of PEPA. This is necessary in order to provide the reward language with a contextual semantics. The rules are presented in the style of Larsen [55]; however, it is assumed that the combinators present in the rules are actually empty contexts. This would mean that the rule given above in equation (3.2.16) can be equated with the form given below.

$$[.] \underset{L}{\bowtie} [.] \xrightarrow[\langle(\alpha, r_1), (\alpha, r_2)\rangle]{(\alpha, R)} [.] \underset{L}{\bowtie} [.] \tag{3.2.17}$$

Notice that all the rules are axioms. It is assumed that the traditional operational semantics of PEPA processes is extended with a *zero* transition, such that

$$P \xrightarrow{0} Q \text{ if and only if } P \equiv Q$$

and where 0 is a distinguished 'zero' activity whose type is not a member of $\mathcal{A}$. The transduction rule for **Prefix** states that no internal activities are consumed in generating an external activity (α, r), and the resulting context is the *identity* context.

	Transition Semantics	**Transduction Semantics**
Prefix	$$\frac{}{(\alpha, r).E \xrightarrow{(\alpha,r)} E}$$	$$\frac{}{(\alpha, r).\ \xrightarrow[0]{(\alpha,r)} I}$$ $$\frac{}{I \xrightarrow[x]{x} I}$$
Choice	$$\frac{E \xrightarrow{(\alpha,r)} E'}{E + F \xrightarrow{(\alpha,r)} E'}$$	$$\frac{}{+ \xrightarrow[\langle(\alpha,r),0\rangle]{(\alpha,r)} \Pi^1} \qquad \frac{}{\Pi^1 \xrightarrow[\langle x,0\rangle]{x} \Pi^1}$$
	$$\frac{F \xrightarrow{(\alpha,r)} F'}{E + F \xrightarrow{(\alpha,r)} F'}$$	$$\frac{}{+ \xrightarrow[\langle 0,(\alpha,r)\rangle]{(\alpha,r)} \Pi^2} \qquad \frac{}{\Pi^2 \xrightarrow[\langle 0,x\rangle]{x} \Pi^2}$$
Coop	$$\frac{E \xrightarrow{(\alpha,r)} E'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E' \underset{L}{\bowtie} F}\ (\alpha \notin L)$$	$$\frac{}{\underset{L}{\bowtie} \xrightarrow[\langle(\alpha,r),0\rangle]{(\alpha,r)} \underset{L}{\bowtie}}\ (\alpha \notin L)$$
	$$\frac{F \xrightarrow{(\alpha,r)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E \underset{L}{\bowtie} F'}\ (\alpha \notin L)$$	$$\frac{}{\underset{L}{\bowtie} \xrightarrow[\langle 0,(\alpha,r)\rangle]{(\alpha,r)} \underset{L}{\bowtie}}\ (\alpha \notin L)$$
	$$\frac{E \xrightarrow{(\alpha,r_1)} E' \quad F \xrightarrow{(\alpha,r_2)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,R)} E' \underset{L}{\bowtie} F'}\ (\alpha \in L)$$	$$\frac{}{\underset{L}{\bowtie} \xrightarrow[\langle(\alpha,r_1),(\alpha,r_2)\rangle]{(\alpha,R)} \underset{L}{\bowtie}}\ (\alpha \in L)$$

where $R = (r_1/r_\alpha(E))(r_2/r_\alpha(F)) \min(r_\alpha(E), r_\alpha(F))$

Hide	$$\frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\alpha,r)} E'/L}\ (\alpha \notin L)$$	$$\frac{}{/L \xrightarrow[(\alpha,r)]{(\tau,r)} /L}\ (\alpha \in L)$$
	$$\frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\tau,r)} E'/L}\ (\alpha \in L)$$	$$\frac{}{/L \xrightarrow[(\alpha,r)]{(\alpha,r)} /L}\ (\alpha \notin L)$$
Const	$$\frac{E \xrightarrow{(\alpha,r)} E'}{A \xrightarrow{(\alpha,r)} E'}\ (A \stackrel{\mathrm{def}}{=} E)$$	$$\frac{}{A \xrightarrow[(\alpha,r)]{(\alpha,r)} E}\ (A \stackrel{\mathrm{def}}{=} E)$$

Figure 3.1: Transduction semantic rules for the combinators of PEPA

This is the context which simply maps internal consumption directly to external production. In Figure 3.1, x is used to range over all activities in $\mathcal{A}ct$, along with the distinguished activity 0. The first transduction rule for **Choice** states that a choice prefix consumes an activity produced by its left operand, and a zero activity (i.e. no activity) from its right operand, produces the left operand's activity for external consumption, and then becomes the *left projection* of a *pair* context. The pair context, Π, is a useful construction that allows the rules given next to be presented simply. A populated pair context possesses two subcomponents, and does not directly represent a process. It is never used on its own—instead, either the first or second projection is used, where $\Pi^1(P_1, P_2) = P_1$ and $\Pi^2(P_1, P_2) = P_2$. Finally, the rule for **Const** transforms a *nullary* context, that is a PEPA variable, into what may be a compound context, by simply mirroring the definitional equation for that variable. The variable context produces only and exactly any activity it consumes.

Larsen shows how the operational semantics of a process algebra are expressible in terms of the operational semantics of the component processes, together with the transduction semantics of contexts. Consider $c[P_1, \ldots, P_n]$, representing a full context (equivalently a PEPA process), where c is a context with n holes. It is stated, without proof, that the transitions of combined processes may be completely characterised in terms of the behaviours of P_1 to P_n, and the transduction semantics of c, by a *uniform rule*:

$$\frac{P_1 \xrightarrow{\mathsf{a_1}} P_1' \ldots P_n \xrightarrow{\mathsf{a_n}} P_n' \quad c \xrightarrow[\langle \mathsf{a_1} \ldots \mathsf{a_n} \rangle]{\mathsf{a}} c'}{c[P_1, \ldots, P_n] \xrightarrow{\mathsf{a}} c'[P_1', \ldots, P_n']} \tag{3.2.18}$$

This rule is generalised in order to describe compositionally the transduction semantics of a context. Consider the context $c[c_1, \ldots, c_n]$, that is the context c populated with subcontexts c_1 to c_n. The *Generalised Uniform Rule* below defines the semantics of $c[c_1, \ldots, c_n]$ in terms of the semantics of c and c_1 to c_n.

$$\frac{c_1 \xrightarrow[\underline{\mathsf{b_1}}]{\mathsf{a_1}} c_1' \ldots c_n \xrightarrow[\mathsf{b_n}]{\mathsf{a_n}} c_n' \quad c \xrightarrow[\langle \mathsf{a_1} \ldots \mathsf{a_n} \rangle]{\mathsf{a}} c'}{c[c_1, \ldots, c_n] \xrightarrow[\underline{b_1} \cdots \underline{b_n}]{\mathsf{a}} c'[c_1', \ldots, c_n']} \tag{3.2.19}$$

Now, a *contextual transition relation* is defined which captures the notion of subcomponents proceeding in context. This transition relation will be used implicitly in the semantics of the Reward language, and so is restricted to non dynamic contexts only.

Definition 3.2.9 (Contextual transition relation). *A* contextual transition relation *relates two equal-length sequences of PEPA processes and two contexts. Let $\underline{P}$ be a length-n vector of PEPA processes, and let c be a non dynamic context with n holes. Then $\underline{P} \xrightarrow{(\alpha,r)}_{c,c'} \underline{P}'$ if and only if $c \xrightarrow[\underline{v}]{(\beta,r)} c'$ for some β, $\underline{v} \neq \underline{0}$, and for $1 \leq i \leq n$, either*

- $v_i = 0$, *or*
- $v_i = (\alpha, r_i)$ *and* $P_i \xrightarrow{(\alpha,r_i)} P'_i$ *for some* r_i

The contexts decorating the transition arrow record the context from which the transition was taken, and the context which results. Of course the resulting context is not uniquely determined, but by Lemma 3.2.4, the existence of a context c' with n holes is guaranteed. The intuition behind this definition is that a subsequence of subcomponents may make transitions *in context* if the context itself may produce a transition via an activity with the correct *rate*, by consuming 0 or more activities from its component processes, each of which must have the correct *action type*. The definition uses the action types of the component processes because the modeller will wish to study the behaviour of subcomponents *in context*, and therefore will write any behavioural specifications in terms of the action types presented by the subcomponents. However the definition incorporates the rate of the activity produced by *the context*, because the context may combine the rates of the subcomponents' activities to produce its composite, and the modeller would wish to study the *performance* of the subcomponents within the model as a whole. To disregard the rate of the context's produced activity would be to disregard, for the most part, the context itself. Notice that a subcomponent may be defined so as to make a transition to itself, for example $P \stackrel{\text{def}}{=} (\alpha, r).P$. Then, even though P may appear unchanged on the right hand side of a contextual transition, it may still have been used in the inference of the transition of the populated context. The definition also insists that any contextual transition must involve *at least one* of the subcomponents. $\underline{P} \xrightarrow{(\alpha,r)}_c$ implies that $\underline{P} \xrightarrow{(\alpha,r)}_{c,c'} \underline{P'}$ for some $\underline{P}, c'$.

Example 2. *Let* $L = \{\alpha, \beta\}$ *and* $c = (P \bowtie_L [.]) \mid\mid Q$. *Next, some simple PEPA processes:* $P \stackrel{\text{def}}{=} (\beta, r).P$, $Q \stackrel{\text{def}}{=} (\alpha, s).Q$, *and* $R \stackrel{\text{def}}{=} (\beta, \top).R + (\alpha, s).R$. *Then*

$$\begin{array}{rcll}
R & \xrightarrow{(\beta,\top)} & R & \\
R & \xrightarrow{(\alpha,s)} & R & \\
\langle R \rangle & \xrightarrow{(\beta,r)}_{c,c} & \langle R \rangle & \\
\langle R \rangle & \not\xrightarrow{(\alpha,t)}_{c,c} & \langle R \rangle & \textit{for any } t \\
c[R] & \xrightarrow{(\alpha,s)} & c[R] &
\end{array} \tag{3.2.20}$$

Notice that the action type decorating a contextual transition is the same as the type decorating the transitions of the subcomponents, but not necessarily the same as the type of the activity produced by the process represented by the populated context. However, if the types are not equal, the type produced by the process represented by the populated context must be the hidden type, τ. This simple fact is stated without proof, and is because PEPA possesses no 'relabelling' combinator. Notice that the same type decorates every subcomponent transition. This is because if several subcomponents proceed in context, the only possibility is due to cooperation. If a PEPA cooperation enables an activity of type α present in its cooperation set, then both subcomponents must also enable an activity of type α. For a PEPA process within a context, the only way in which the witnessed activity may not be of type α is if the cooperation context which produces the activity is within the scope of a hiding context, where the hiding set contains the type α. This may not be true for more ornate Markovian process algebras, but the theory presented here should nevertheless be broadly applicable.

As described, the rate decorating a contextual transition is different from the rates of the subcomponent transitions, in general. It reflects the capacity of the whole model to perform the transition, and not the capacity of any particular subcomponent. The motivation is again reflected in the needs of the PEPA modeller, who may wish to study the behavioural properties of a subcomponent within a context, 'ignoring' the activities of other subcomponents. Within a context, a subcomponent may be required to perform activities at rates mandated by the cooperation rule. Therefore, it makes no sense to examine the subcomponent's individual transition rates, otherwise the context serves little purpose.

The following lemma is useful to the results which follow. It investigates a subcontext c' of c where c' is empty (intuitively, all 'leaves' of c' are holes) and non dynamic. The result states that if c' is fully populated such that it is strongly equivalent to P, then the contextual transitions of a fully populated c are identical to those of a similarly populated c except where c' is replaced by P.

Lemma 3.2.5. *Let $P, P_1, \dots, P_n$ be PEPA processes, and c, c', c'' be contexts such c' is empty and non dynamic, $P \cong c'[P_1, \dots, P_n]$, and $c[c'] = c''$.*

Then $\langle P\rangle \xrightarrow{(\alpha,r)}_{c,d} \langle P'\rangle$ if and only if $\langle P_1, \dots, P_n\rangle \xrightarrow{(\alpha,r)}_{c'',d''} \langle P'_1, \dots, P'_n\rangle$ where $d[c'] = d''$ and $P' \cong c'[P'_1, \dots, P'_n]$.

Proof. Assume that $\langle P\rangle \xrightarrow{(\alpha,r)}_{c,d} \langle P'\rangle$. Then $c \xrightarrow[(\alpha,r)]{(\beta,r)} d$ for some β, and $P \xrightarrow{(\alpha,r)} P'$ by Definition 3.2.9. But by the definition of strong equivalence, it is then the case that $c'[P_1, \dots, P_n] \xrightarrow{(\alpha,r)} c'[P'_1, \dots, P'_n]$, since c' is empty and non dynamic. Now by the uniform rule (3.2.18), it is the case that $c' \xrightarrow[\underline{v}]{(\alpha,r)} c'$, and for $1 \leq i \leq n$, $P_i \xrightarrow{v_i} P'_i$. Further, since $c \xrightarrow[(\alpha,r)]{(\beta,r)} d$ by assumption, and $c' \xrightarrow[\underline{v}]{(\alpha,r)} c'$, then the generalised uniform rule (3.2.19) may be applied leading to $c[c'] \xrightarrow[\underline{v}]{(\beta,r)} d[c']$. However $c[c'] = c''$ and $d[c'] = d''$, and so by definition, $\langle P_1, \dots, P_n\rangle \xrightarrow{(\alpha,r)}_{c'',d''} \langle P'_1, \dots, P'_n\rangle$. The reverse direction may be proved by similar reasoning. □

Now that a contextual transition relation has been defined, Definition 3.2.12 makes concrete what is meant by the satisfaction of a PML$_\mu$ formula in a context. First extensions of two earlier definitions are given.

Definition 3.2.10. $\mu_{\underline{P},c,\alpha}(\underline{P}', c') = \sum\{\!|r : \underline{P} \xrightarrow{(\alpha,r)}_{c,c'} \underline{P}'|\!\}$

Definition 3.2.11. *Let $\underline{P}$ be a length-n vector of PEPA models and c be a context with n holes. Let S be a set of pairs, each of which is a length-n vectors of processes, and a context with n holes. Then $\underline{P} \stackrel{(\alpha,\nu)}{\Longrightarrow}_c S$ if and only if $\underline{P} \xrightarrow{(\alpha,r)}_c$ for some r, and $\sum_{(\underline{P}',c') \in S} \mu_{\underline{P},c,\alpha}(\underline{P}', c') = \nu$*

Set S consists of pairs of sequences of subcomponents and contexts. If a subcomponent sequence and its context are present in S, it is because this context can be reached via a contextual transition from $\underline{P}$. Notice that this is still a one-step relation—the context emits one activity in response to a set of activities produced by its component processes. This condition is important otherwise the result would be misleading. To see this, consider the following definitions:

$$\begin{aligned}
P &\stackrel{\text{def}}{=} (\alpha, \top).P' \\
Q_1 &\stackrel{\text{def}}{=} (\alpha, r_1).Q' + (\beta, s).Q_2 \\
Q_2 &\stackrel{\text{def}}{=} (\alpha, r_2).Q' \\
c_1 &= [.] \underset{\{\alpha\}}{\bowtie} Q_1 \\
c_2 &= [.] \underset{\{\alpha\}}{\bowtie} Q_2 \\
d &= [.] \underset{\{\alpha\}}{\bowtie} Q'
\end{aligned} \tag{3.2.21}$$

The context c_1 populated by P may make a transition via activity (α, r_1) to a process represented by the context d populated by P'. Alternatively, c_1 may make a transition via (β, s) to context c_2 without involving P, and then the first transition may be made but instead from c_2 to d. However this does *not* mean that P has the capacity to perform α at rate $r_1 + r_2$. The one-step definition ensures there is no confusion.

Definition 3.2.12 (PML$_\mu$ **satisfaction in context).** *Let P_i be a PEPA model, for $1 \leq i \leq n$, and $\underline{P} = \langle P_1, \ldots, P_n \rangle$. Let c be a context with n holes. Then $\models_c$ is a* satisfaction relation *defined as the least relation satisfying the following rules:*

$\underline{P} \models_c \texttt{tt}$

$\underline{P} \models_c \neg F$ *if* $\underline{P} \not\models_c F$

$\underline{P} \models_c F_1 \wedge F_2$ *if* $\underline{P} \models_c F_1$ *and* $\underline{P} \models_c F_2$

$\underline{P} \models_c \nabla_\alpha$ *if* $\underline{P} \overset{(\alpha,r)}{\not\longrightarrow}_{c,c'}$ *for any* c' *or* r

$\underline{P} \models_c \langle\alpha\rangle_\mu F$ *if* $\exists n \in \mathbb{N}$ *such that* $\underline{P} \models_c \langle\alpha\rangle^n_\mu F$

$$\underline{P} \models_c \langle\alpha\rangle^n_\mu F \textit{ if either } \begin{cases} \underline{P} \overset{(\alpha,\nu)}{\Longrightarrow}_c S \textit{ for some } \nu \geq \mu, \textit{and for all } (\underline{P}', d) \in S, \underline{P}' \models_d F \\ c \xrightarrow[\underline{0}]{\texttt{a}} c' \textit{ for some } \texttt{a}, n > 0, \textit{ and } \underline{P} \models_{c'} \langle\alpha\rangle^{n-1}_\mu F \end{cases}$$

A new formula, $\langle\alpha\rangle^n_\mu F$, has been introduced here, for the purposes of presentation, and not for use by the PEPA modeller. It is satisfied by a vector of processes in context if the context makes no more than n transitions without involving its subcomponents, before a context is reached in which the subcomponents may make a number of different α-transitions such that the aggregate rate of departure from the context is at least μ. This prevents problems where the context may cycle infinitely often. In practise, this is not a restriction, since the modeller will be working with finite-state models; there will never be a situation where the modeller would expect a formula of the form $\langle\alpha\rangle^n_\mu F$ to be true, but only after an infinite number of contextual transitions which do not involve the subcomponents.

Example 3. *Consider the following PEPA model:*

$$\begin{aligned} \mathit{Client} &\overset{\text{def}}{=} (\texttt{conn}, \lambda).(\texttt{disc}, \mu).\mathit{Client} \\ \mathit{Server} &\overset{\text{def}}{=} (\texttt{conn}, \top).(\texttt{serve}, s).\mathit{Server} + (\texttt{disc}, \top).\mathit{Server} \\ \mathit{Network} &\overset{\text{def}}{=} (\mathit{Client} \parallel \mathit{Client}) \underset{\{\texttt{conn},\texttt{disc}\}}{\bowtie} \mathit{Server} \end{aligned} \tag{3.2.22}$$

The PEPA modeller may only be interested in the behaviour of the Client subcomponents. Thus, let $c = ([.] \parallel [.]) \underset{\{\texttt{conn},\texttt{disc}\}}{\bowtie} \mathit{Server}$*, and* $d = ([.] \parallel [.]) \underset{\{\texttt{conn},\texttt{disc}\}}{\bowtie} (\texttt{serve}, s).\mathit{Server}$*. Now, for a particular derivative of the model, the modeller may wish to determine if neither client has currently established a connection to the server. By inspection of Client only, no connection is currently established if* $\mathit{Client} \overset{\texttt{conn}}{\longrightarrow}$*. Therefore, neither client has a connection if*

$$\langle \mathit{Client}, \mathit{Client} \rangle \models_c \langle\texttt{conn}\rangle_\lambda \langle\texttt{conn}\rangle_\lambda \texttt{tt}$$

This formula is satisfied despite the fact that Server must perform some observable work between connections. The above claim is proved with the following steps:

1. $\langle(\texttt{disc}, \mu).\mathit{Client}, (\texttt{disc}, \mu).\mathit{Client}\rangle \models_c \texttt{tt}$ *is true trivially.*

2. • $d \xrightarrow[\underline{0}]{(\mathtt{serve},s)} c$, *and*

• $\langle(\mathtt{disc},\mu).\mathit{Client},\mathit{Client}\rangle \overset{(\mathtt{conn},\lambda)}{\Longrightarrow}_{c} \{(\langle(\mathtt{disc},\mu).\mathit{Client},(\mathtt{disc},\mu).\mathit{Client}\rangle, c)\}$, *therefore,*

• $\langle(\mathtt{disc},\mu).\mathit{Client},\mathit{Client}\rangle \models_d \langle\mathtt{conn}\rangle^1_\lambda\mathtt{tt}$ *by definition, and so*

• $\langle(\mathtt{disc},\mu).\mathit{Client},\mathit{Client}\rangle \models_d \langle\mathtt{conn}\rangle_\lambda\mathtt{tt}$ *by definition.*

3. • $\langle\mathit{Client},\mathit{Client}\rangle \overset{(\mathtt{conn},\lambda)}{\Longrightarrow}_{c} \{(\langle(\mathtt{disc},\mu).\mathit{Client},\mathit{Client}\rangle, d)\}$, *therefore,*

• $\langle\mathit{Client},\mathit{Client}\rangle \models_c \langle\mathtt{conn}\rangle^0_\lambda\langle\mathtt{conn}\rangle_\lambda\mathtt{tt}$ *by definition, and so*

• $\langle\mathit{Client},\mathit{Client}\rangle \models_c \langle\mathtt{conn}\rangle_\lambda\langle\mathtt{conn}\rangle_\lambda\mathtt{tt}$ *by definition.*

Finally in this section, a result is presented which highlights the extent to which the PEPA modeller may employ aggregation while focusing on subcomponents of a model. Intuitively, it states that it is safe to aggregate subcomponents which are within the view of an empty context. That is, the truth of a PML_μ formula in context is preserved if subcomponents in an empty context are aggregated.

Theorem 3.2.2 (Aggregation preserved formulas). *Let $P, P_1, \ldots, P_n$ be PEPA processes, and c, c', c'' be contexts such that c' is empty, non dynamic, and contains no subcontext d/L, $P \cong c'[P_1, \ldots, P_n]$, and $c[c'] = c''$ Then for any PML_μ formula F,*

$$\langle P\rangle \models_c F \text{ if and only if } \langle P_1, \ldots, P_n\rangle \models_{c''} F$$

Proof. The proof is by induction over the structure of F.

Cases $F = \mathtt{tt}, F = G \wedge H, F = \neg G, F = \nabla_\alpha$: Straightforward.

Case $F = \langle\alpha\rangle_\mu G$: Assume that $\langle P\rangle \models_c \langle\alpha\rangle_\mu G$. Then $\langle P\rangle \models_c \langle\alpha\rangle^n_\mu G$ for some $n > 0$. This implies that at least one of the following two sub-cases is true.

• $c \xrightarrow[0]{\mathtt{a}} d$, $\langle P\rangle \models_d \langle\alpha\rangle^{n-1}_\mu G$. Then by induction, $\langle P_1, \ldots, P_n\rangle \models_{d''} \langle\alpha\rangle^{n-1}_\mu G$ where $d[c'] = d''$. Now, by the generalised uniform rule (3.2.19), the following can be shown:

$$\frac{c' \xrightarrow[0]{0} c' \quad c \xrightarrow[0]{\mathtt{a}} d}{c[c'] \xrightarrow[0]{\mathtt{a}} d[c']}$$

Therefore, $c'' \xrightarrow[0]{\mathtt{a}} d''$. Now, by definition, it is the case that $\langle P_1, \ldots, P_n\rangle \models_d \langle\alpha\rangle^n_\mu G$, and therefore, $\langle P_1, \ldots, P_n\rangle \models_d \langle\alpha\rangle_\mu G$.

- $\langle P\rangle \overset{(\alpha,\nu)}{\Longrightarrow}_c S$, $\nu \geq \mu$, and for all $(\langle P'\rangle, d) \in S, \langle P'\rangle \models_d G$. Then for all $(\langle P'\rangle, d) \in S$, by definition, $\langle P\rangle \xrightarrow{(\alpha,\nu)}_{c,d} \langle P'\rangle$; and therefore $c \xrightarrow[(\alpha,\nu)]{(\beta,\nu)} d$, and $P \xrightarrow{(\alpha,\nu)} P'$. But then by the definition of strong equivalence, $c[P_1, \ldots, P_n] \xrightarrow{(\alpha,\nu)} c'[P'_1, \ldots, P'_n]$ (the existence of suitable $P'_1, \ldots, P'_n$ is guaranteed by Lemma 3.2.4), and $c'[P'_1, \ldots, P'_n] \cong P'$. By examination of the inference tree for $c[P_1, \ldots, P_n] \xrightarrow{(\alpha,\nu)} c'[P'_1, \ldots, P'_n]$, and since c' is empty and contains no subcontext of the form d/L, it can be seen that for $1 \leq i \leq n$, either $P_i \xrightarrow{(\alpha,\nu_i)} P'_i$ for some ν_i, or P_i does not contribute, and $\sum_i \nu_i = \nu$. Let $v_i = (\alpha, \nu_i)$ if P_i contributes, and 0 if it does not; then $c' \xrightarrow[\underline{v}]{(\alpha,\nu)} c'$. Now by the generalised uniform rule (3.2.19), the following can be shown:

$$\frac{c' \xrightarrow[\underline{v}]{(\alpha,\nu)} c' \quad c \xrightarrow[(\alpha,\nu)]{(\beta,\nu)} d}{c[c'] \xrightarrow[\underline{v}]{(\beta,\nu)} d[c']}$$

  Therefore, $c'' \xrightarrow[\underline{v}]{(\beta,\nu)} d''$. Now, by definition, $\langle P_1, \ldots, P_n\rangle \xrightarrow{(\alpha,\nu)}_{c'',d''} \langle P'_1, \ldots, P'_n\rangle$, where $P_i \equiv P'_i$ if $v_i = 0$. However, since $d[c'] = d''$, and $c'[P'_1, \ldots, P'_n] \cong P'$, by induction it is the case that $\langle P'_1, \ldots, P'_n\rangle \models_{d''} G$. Since the above derivation was worked out for all $(\langle P'\rangle, d) \in S$, then it is simple to construct an appropriate S'' such that $\langle P_1, \ldots, P_n\rangle \overset{(\alpha,\nu)}{\Longrightarrow}_{c''} S''$ and for all $(\underline{P}', d'') \in S''$, $\langle P'_1, \ldots, P'_n\rangle \models_{d''} G$.

This provides a proof of the forward direction of the implication; the reverse direction can be proved by similar reasoning. □

On the other hand, it is simple to demonstrate that PML$_\mu$ formulas in context are not necessarily preserved when the aggregated subcontext is not empty.

Example 4. *Consider the following definitions:*

$$\begin{aligned}
P &\stackrel{\text{def}}{=} (\alpha, 2r).P' & Q &\stackrel{\text{def}}{=} (\alpha, r).Q' \\
P' &\stackrel{\text{def}}{=} (\beta, s).P + (\alpha, r).P'' & Q' &\stackrel{\text{def}}{=} (\beta, s).Q \\
P'' &\stackrel{\text{def}}{=} (\beta, 2s).P' & c' = c'' &= [.] \mid\mid Q \\
c &= [.] & d &= [.] \mid\mid Q' \\
F &= \langle\alpha\rangle_{3r/2}\mathtt{tt}
\end{aligned} \tag{3.2.23}$$

Each context presented is non dynamic, and c' is not empty. The following derivations *show that despite the fact that $P \cong c'[Q]$, the satisfaction of F depends on the context.*

$$\begin{aligned}
& P \xrightarrow{(\alpha,2r)} P' \\
\Longrightarrow\ & c[P] \xrightarrow{(\alpha,2r)} c[P'] \\
\Longrightarrow\ & \langle P \rangle \xrightarrow{(\alpha,2r)}_{c,c} \langle P' \rangle \\
\Longrightarrow\ & \langle P \rangle \stackrel{(\alpha,2r)}{\Longrightarrow}_{c} \{(\langle P' \rangle, c)\} \\
\Longrightarrow\ & \langle P \rangle \models_c \langle \alpha \rangle_{3r/2} \texttt{tt} \\
\Longrightarrow\ & \langle P \rangle \models_c F
\end{aligned} \tag{3.2.24}$$

However, in order that $\langle Q \rangle \models_{c''} F$, it must be the case that $\langle Q \rangle \stackrel{(\alpha,\nu)}{\Longrightarrow}_{c''} S$ for some set S. However it is only possible to deduce

$$\begin{aligned}
& \langle Q \rangle \stackrel{(\alpha,r)}{\Longrightarrow}_{c''} \{(\langle Q' \rangle, c'')\} \\
& \langle Q \rangle \stackrel{(\alpha,r)}{\Longrightarrow}_{d} \{(\langle Q' \rangle, d)\}
\end{aligned} \tag{3.2.25}$$

and $c'' \neq d$.

This is consistent with our intuition—the context c' restricts the modeller's view of the subcomponent's partner. In this example, the satisfaction of F depends crucially on the particular rate chosen. In practice, it is often the case that a modeller wishes to study only the qualitative behaviour of a component. This can be done by restricting all rates in a PML_μ formula to be less than ϵ, the rate due to the minimal rate assumption. Such a PML_μ formula is called *rate-reduced*.

Definition 3.2.13. *Let F be a PML_μ formula. Then F is* rate-reduced *if every subformula of F of the form $\langle \alpha \rangle_\mu G$ is such that $\mu \leq \epsilon$.*

This section ends with the following conjecture on the satisfaction of a rate-reduced PML_μ formula in a context that need not be empty.

Conjecture 1. *Let $P, P_1, \ldots, P_n$ be PEPA processes, and c, c', c'' be contexts such that c' is non dynamic and contains no subcontext d/L, $P \cong c'[P_1, \ldots, P_n]$, and $c[c'] = c''$. Then for any* rate-reduced PML_μ *formula F,*

$$\langle P \rangle \models_c F \textit{ if and only if } \langle P_1, \ldots, P_n \rangle \models_{c''} F$$

This result would imply that a modeller may aggregate a model arbitrarily, and guarantee that the satisfaction of any such formula is unaffected.

3.3 Description of the PEPA Reward Language

In this section, a formal description of the PEPA Reward language is given, including a simple syntax and semantics. Using the Reward language described here, the method 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 with which process derivatives a particular reward specification is associated.

The meaning of the reward specification will depend on how it is 'attached' to a PEPA model; this is because the associated value may depend on information local to the derivative under consideration. This will be explained when the semantics of the Reward language is described in Section 3.3.1.

Formally, each reward specification can be considered as a pair consisting of a logical formula and a reward expression. The formula is checked against a set of subcomponents in a particular chosen context. If satisfied, the derivative equivalent to that given by placing each subcomponent in context is assigned a reward. The value of the reward corresponds to the evaluation of a simple arithmetic-like expression.

3.3.1 Syntax and Semantics of Reward Expressions

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

$$\begin{array}{rcl} e & ::= & (e) \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1/e_2 \mid \mathsf{atom} \\ \mathsf{atom} & ::= & r \in \mathbb{R} \mid cur \mid rate(\alpha \in \mathcal{A}) \end{array}$$

The bound variables cur and $rate()$ will be used to denote real numbers. The meaning will be dependent on the reward structure being built, 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. These simple expressions are given a semantics below. Because the meaning of one of the bound variables depends on activity rates, the semantic function is decorated with information from the transition system.

First, a reward function is defined. If P is the PEPA model under consideration, then let $\rho(\cdot) : ds(P) \longrightarrow \mathbb{R}$. Therefore, given a derivative (in fact a state of the transition system) $\rho(\cdot)$ gives the reward assigned to that derivative. Given this reward assignment function, the semantic function relies on a view of a PEPA process P to define the meaning of reward expressions; the semantics are given in Figure 3.2. The variable *cur* is

$$\begin{array}{rcl}
\| (e) \|_{(\underline{P},c)} & = & \| e \|_{(\underline{P},c)} \\
\| e_1 \ op \ e_2 \|_{(\underline{P},c)} & = & \| e_1 \|_{(\underline{P},c)} \ op \ \ \| e_2 \|_{(\underline{P},c)} \\
\| cur \|_{(\underline{P},c)} & = & \rho(c[\underline{P}]) \\
\| rate(\alpha) \|_{(\underline{P},c)} & = & \sum\{r : c[\underline{P}] \xrightarrow{(\alpha,r)}\}
\end{array}$$

Figure 3.2: Semantics of reward expressions

intended to give the reward expression access to a 'currently' assigned reward, allowing reward expressions to make use of previous assignments. 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. This is the way in which timing information may be incorporated into reward specifications. Notice that the semantics insist the rate is that of α as performed by the model, not by a subcomponent in context. This is because for any given view of a context, it may be possible to witness a contextual transition via an α at a number of different rates. It is not clear that it is correct to choose any one of these rates, and the current semantics is unambiguous. The binary operator *op* is intended to capture the obvious binary operators defined in the syntax above. The following definition completes the definition of a reward specification.

Definition 3.3.1. *A* reward specification *is a pair (F, e), where F is a* PML$_\mu$ *formula and e is a reward expression.*

Notice that the semantics of a reward expression is a function which satisfies Equation (3.2.10), assuming use of a 'prior' reward function $\rho(\cdot)$ which satisfies Equation (3.2.10).

Lemma 3.3.1. *Let $(\underline{P}, c)$ be a view of a PEPA process, $c'[\underline{P}'], c''[\underline{P}''] \in ds(c[\underline{P}])$ such that $c'[\underline{P}'] \cong c''[\underline{P}'']$, and $\rho(\cdot)$ be a reward function such that $\rho(c'[\underline{P}']) = \rho(c''[\underline{P}''])$. Then for any reward expression e, $\| e \|_{(\underline{P}',c')} = \| e \|_{(\underline{P}'',c'')}$.*

Proof. The proof is by induction over the structure of e, and is straightforward—the interesting cases are shown.

Case $e = cur$: Then $\| e \|_{(\underline{P}',c')} = \rho(c'[\underline{P}']) = \rho(c''[\underline{P}'']) = \| e \|_{(\underline{P}'',c'')}$.

Case $e = rate(\alpha)$: Then $\| e \|_{(\underline{P}',c')} = \sum\{r : c'[\underline{P}'] \xrightarrow{(\alpha,r)}\}$. By the definition of strong equivalence, this is equal to $\sum\{r : c''[\underline{P}''] \xrightarrow{(\alpha,r)}\} = \| e \|_{(\underline{P}'',c'')}$.

Other cases : Trivial.

□

3.3.2 Creating a Reward Structure with Attachments

The next task is to use a reward specification to build a reward structure for a given PEPA process. This section defines the semantics of the PEPA Reward language.

The semantics depends on some further definitions. These help capture the idea that rewards can be attached to subcomponents of PEPA processes, thus allowing the compositional structure of the process algebra model to be exploited. For instance, given a large PEPA model, it may be interesting to only examine the performance of a single component queue. It should be possible to disregard the behaviour of the rest of the model, at least up to its interaction with the queue under examination. To achieve this, contexts are employed.

Definition 3.3.2. *An* attachment *is a triple* $(\sigma, c, \langle P_1, \ldots, P_n \rangle)$*, where* σ *is a reward specification,* c *is a non dynamic context with* n *holes, and* P_i *are PEPA processes, for* $1 \leq i \leq n$*.*

The attachment allows the modeller to choose which subcomponents are of interest—the subcomponents are the processes P_i.

Now let $\rho : ds(P) \longrightarrow \mathbb{R}$ represent a function constructing a reward structure, and let P be a PEPA process. Assume an initial value of $\rho(P') = 0$, for all $P' \in ds(P)$. This is chosen arbitrarily—however it is a reasonable and useful choice in practice [66]. The semantic function takes as an argument a reward assignment function ρ and evaluates to a new function, say ρ'. This possibly modified assignment function will reflect any new rewards that have been assigned to the PEPA model. Its argument is a *sequence* of attachments. A sequence is chosen so a reward structure can be built sequentially, allowing one reward expression to make use of the values present in the partially constructed reward structure. Evaluating such a sequence of attachments is trivial—each is evaluated individually, in order. This is shown below.

$$\begin{aligned} \| \langle \rangle \|_\rho &= \rho \\ \| \langle a_i, a_{i+1}, \ldots, a_m \rangle \|_\rho &= \| \langle a_{i+1}, a_{i+2}, \ldots, a_m \rangle \|_{\rho'} \text{ where } \rho' = \| a_i \|_\rho \end{aligned} \tag{3.3.1}$$

The meaning of an attachment is now required.

Definition 3.3.3 (Semantics of an attachment). *Let* $a_i = ((F, e), c, \langle P_1, \ldots, P_n \rangle)$

be an attachment. Then the meaning of an attachment is a value determined as follows:

$$\| a_i \| = \begin{cases} \| e \|_{\langle P_1, \ldots, P_n \rangle_c} & \textit{if } \langle P_1, \ldots, P_n \rangle \models_c F \\ 0 & \textit{otherwise.} \end{cases} \tag{3.3.2}$$

ρ' is created by ordinary function perturbation and the end result is a function which constructs a reward structure over the derivative space of a PEPA process.

3.4 Examples of Reward Specification

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

3.4.1 Abstract Multi-processor Multi-memory Example

A model that occurs frequently in the literature is the *multi-processor multi-memory* system. Such systems have become affordable and even commonplace in recent years, and for these purposes, have the advantage that the process algebra style is a particularly suitable way to represent them. Consider an abstraction of a modern computer system, containing several processors and several memory chips. Each processor performs some work, then attempts to access memory. The individual components of this system can be modelled in the following way:

$$\begin{aligned} \mathit{Mem}_i &\stackrel{\text{def}}{=} (\mathtt{get}_{\mathtt{Mi}}, \top).(\mathtt{rel}_{\mathtt{Mi}}, \top).(\mathtt{refresh}, r).\mathit{Mem}_i \\ \mathit{Proc}_j &\stackrel{\text{def}}{=} (\mathtt{work}, w_j).\Big(\sum_k (\mathtt{get}_{\mathtt{Mk}}, g \cdot p_k).(\mathtt{rel}_{\mathtt{Mk}}, r).\mathit{Proc}_j \\ &\quad + (\mathtt{interrupt}, i_j).(\mathtt{rti}, r_j).\mathit{Proc}_j\Big) \\ &\quad \text{where } \sum_k p_k = 1 \end{aligned} \tag{3.4.1}$$

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. Alternatively, at this point the processor can be interrupted by a peripheral device which it then has to service. To make things more illustrative, this particular multi-processor system will be extended with an 'IO Controller' chip. This processor is only capable of accessing memory chip Mem_1:

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

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

$$\begin{aligned} \mathit{System} \;&\stackrel{\text{def}}{=}\; (\mathit{Proc_1} \parallel \mathit{Proc_2} \parallel \mathit{IOC}) \underset{L}{\bowtie} (\mathit{Mem_1} \parallel \mathit{Mem_2}) \\ &\quad\text{where } L = \{\mathtt{get_{Mi}}, \mathtt{rel_{Mi}} \mid i = 1, 2\} \end{aligned} \tag{3.4.2}$$

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

A Utilisation Measure

In order to determine how many memory chips are needed for reasonable performance, it may be useful 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 $\mathtt{rel_{M1}}$; if so, it previously performed a $\mathtt{get_{M1}}$ in synchrony with a processor which required access. The required reward specification and attachment are then given by

$$\begin{aligned} \text{spec} &= (\Delta_{\mathtt{rel_{M1}}}, 1) \\ \text{attachment} &= (\text{spec}, c, \langle \mathit{Proc_1}, \mathit{Proc_2}, \mathit{IOC}, \mathit{Mem_1}, \mathit{Mem_2} \rangle) \\ &\quad \text{where } c = ([.] \parallel [.] \parallel [.]) \underset{L}{\bowtie} ([.] \parallel [.]) \end{aligned} \tag{3.4.3}$$

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. Nothing more focused is required in this instance. The reward assignment procedure will begin with the process *System*, and will determine whether it is capable of satisfying the logical formula $\Delta_{\mathtt{rel_{M1}}}$. Clearly none of the memory chips are in use at this point, so it will not be possible to release any, and specifically not $\mathit{Mem_1}$. 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* and examine them 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 $\mathit{Mem_1}$ is in use.

The above example would become slightly more complicated if the percentage of time a true 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 and attachment would be:

$$\begin{aligned} \text{spec} &= (\Delta_{\texttt{rel}_{\texttt{M1}}} \wedge \Delta_{\texttt{rel}_{\texttt{M2}}}, 1) \\ \text{attachment} &= (\text{spec}, c, \langle Proc_1, Proc_2, IOC, Mem_1, Mem_2 \rangle) \end{aligned} \tag{3.4.4}$$

since it should be able to use Mem_1 or Mem_2, whichever should become available.

A Performability Measure

While the multi-processor system may give adequate performance for the most part, the modeller may wish to study the effect on the performance if the rate of peripheral interrupts is changed. For example, the modeller may wish to ensure that the rate of access to memory remains above a particular level. To achieve this, the following reward specification may be used:

$$\text{spec} = (\Delta_{\texttt{rel}_{\texttt{M1}}} \implies \langle \texttt{rel}_{\texttt{M1}} \rangle_\mu \texttt{tt} \wedge \Delta_{\texttt{rel}_{\texttt{M2}}} \implies \langle \texttt{rel}_{\texttt{M2}} \rangle_\mu \texttt{tt}, 1) \tag{3.4.5}$$

Here the modeller insists that the rate of access to shared memory is at least μ, for both processors. Using the same attachment, a reward structure will be built which will evaluate to 1 if the condition is always true, and a value less than 1 if the model can reach a state where the rate of access to shared memory drops below the required threshold.

Instead, the modeller may wish to calculate a value corresponding to the throughput of accesses to shared memory. The next specification achieves this.

A Throughput Measure

A 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 again. 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:

$$\begin{aligned} \text{spec} &= (\Delta_{\texttt{work}}, rate(\texttt{work})) \\ \text{attachment} &= (\text{spec}, c', \langle IOC \rangle) \\ \text{where } c' &= (Proc_1 \parallel Proc_2 \parallel [.]) \underset{L}{\bowtie} (Mem_1 \parallel Mem_2) \end{aligned} \tag{3.4.6}$$

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 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 (for example it must cooperate 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.

Illustrating the Logic

In order to illustrate the discriminatory power of the modal logic, a more intricate specification 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 $\mathtt{rel}_{\mathtt{Mi}}$ activity, and its one-step derivative via $\mathtt{rel}_{\mathtt{Mi}}$ enables a $\mathtt{rel}_{\mathtt{Mj}}$ activity. Combining this with the example above, the following reward specification is obtained:

$$\text{spec} \quad = \quad ((\Delta_{\mathtt{rel}_{\mathtt{M1}}} \vee \Delta_{\mathtt{rel}_{\mathtt{M2}}}) \wedge [\mathtt{relM1}]\nabla_{\mathtt{rel}_{\mathtt{M2}}} \wedge [\mathtt{relM2}]\nabla_{\mathtt{rel}_{\mathtt{M1}}}, 1) \tag{3.4.7}$$

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.

3.4.2 A Problematic Example

The PEPA Reward language seems less well-suited to the specification of 'counting' rewards, such as expected queue length. To highlight this, a new model is introduced.

$$\begin{aligned} \mathit{Queue_0} &\stackrel{\text{def}}{=} (\mathtt{arrive}, \lambda).\mathit{Queue_1} \\ \mathit{Queue_i} &\stackrel{\text{def}}{=} (\mathtt{arrive}, \lambda).\mathit{Queue_{i+1}} \\ &+ \; (\mathtt{serve}, \mu).\mathit{Queue_{i-1}} \\ \mathit{Queue_n} &\stackrel{\text{def}}{=} (\mathtt{serve}, \mu).\mathit{Queue_{n-1}} \end{aligned} \tag{3.4.8}$$

This models a simple n-place queue which accepts customers while not full, and serves customers while not empty. Frequently it is useful to know a measure such as the average number of customers in the queue. Thinking behaviourally, there are m customers in the queue if it is possible for the queue to perform m `serve` activities, but not more (alternatively, if it is possible for the queue to perform $n - m$ `arrive` activities but not more). Using the Reward language, it is possible to assign a reward of m to derivatives representing the case where m customers are in the queue:

$$\text{spec} = (\overbrace{\langle\texttt{serve}\rangle \ldots \langle\texttt{serve}\rangle}^{m \text{ times}}\,\texttt{tt} \wedge \overbrace{[\texttt{serve}] \ldots [\texttt{serve}]}^{m+1 \text{ times}}\,\texttt{ff}, m)$$

However what is required is to assign a varying reward of m to a derivative dependent on the number of `serve` activities possible in a row. The Reward language is able to support such reasoning with the same reward expression, by using multiple specifications and attachments, as illustrated below.

$$\begin{array}{rcl}
\text{spec}_1 & = & (\langle\texttt{serve}\rangle\texttt{tt}, \mathit{cur} + 1) \\
\vdots & & \vdots \\
\text{spec}_n & = & (\overbrace{\langle\texttt{serve}\rangle \ldots \langle\texttt{serve}\rangle}^{n \text{ times}}\,\texttt{tt}, \mathit{cur} + 1) \\
\text{attachment}_1 & = & (\text{spec}_1, \mathit{Queue}_0, 1) \\
\vdots & & \vdots \\
\text{attachment}_n & = & (\text{spec}_n, \mathit{Queue}_0, 1)
\end{array} \qquad (3.4.9)$$

where attachment$_i$ is to be evaluated before attachment$_j$ if $i < j$. For each attachment$_i$, every derivative of Queue_0 is checked against the formula contained in spec$_i$. The formula is true for a derivative if it represents a state where there are *at least* i customers. The reward expression adds 1 to the reward previously assigned to the derivative—this would be assigned by the previous attachment, if the state represented the presence of at least $i - 1$ customers. By an inductive argument, this would correctly reward a state with the number of customers it represented. However, a naïve look at the calculation of this reward suggests that for each derivative of Queue_0, the satisfaction of n formulas, each of length $O(n)$, must be checked. This approach is unsatisfactory.

An alternative may be to allow reward expressions to refer to *other* derivatives, for example those to which it can make, or be reached by, a single transition. An example reward expression could look like:

$$\text{newspec} = (\langle\texttt{arrive}\rangle\texttt{tt}, \mathit{prev} + 1)$$

The idea is that if the 'previous' derivative represented a state in which i customers were present, and the current derivative is capable of a further `arrive` activity, then the current derivative represents a state where $i + 1$ customers are queueing. This assumes that the current derivative was reached by a transition signifying the arrival of a customer, and such a notion would need to be made precise. Moreover, any extension to reward expressions would need to satisfy Lemma 3.3.1, and it is not clear that this technique would.

3.5 A Partial Implementation of the PEPA Reward Language

The PEPA Workbench has been extended to allow the use of a subset of the PEPA Reward language. This gives the modeller the capability to express behavioural properties using PML_μ, though currently the use of contexts is not implemented. The implementation automatically generates a reward structure which provably generates the same performance measures for any two strongly equivalent models. This means the modeller may apply aggregation to a PEPA model without having to alter the description of any performance measures.

Given a PEPA model, the Workbench currently generates a representation of the model's generator matrix. This matrix is then solved by a small program implementing the biconjugate gradient algorithm. In order to generate the matrix, it is necessary for the Workbench to traverse the entire state space of the PEPA model. After this traversal, for each state of the model, a reward specification can be checked, and if satisfied, a reward assigned.

The syntax is demonstrated in Figure 3.3 with a simple throughput analysis presented in [18]. *Dev* attempts to transmit a packet consisting of a connection header and some data. However it may fail to connect, in which case it immediately retries. The modeller may wish to consider only the throughput of legitimate data, and not failed connection attempts. The logical expression, named `rate-trans`, is only satisfied in states which enable an activity of type `trans`, but not those states from which two `trans` activities are possible consecutively. Therefore the logical expression selects only those states in which *Dev* has made a connection, and is about to transmit data. These states are assigned the data transmission rate as a reward. The six-state model has a reward assigned to states 3 and 6 only.

The algorithm used to implement this subset of the Reward language employs a simple model checking procedure for PML_μ. The implementation language is Standard ML, and a (sanitised) representative excerpt is given in Figure 3.4. The algorithm is parameterised by a PML_μ formula F, and a derivative of a PEPA process, P. The propositional cases are simply dealt with recursively. The most interesting case is for F of the form $\langle\alpha\rangle_\mu G$, that is `Dia(a,mu,F)`. In order to test the satisfaction of such a

```
% L = { check } ;

# Dev = (check, r1).Try ;
# Try = (trans, fail).Try + (trans, succeed).Con ;
# Con = (trans, data).Dev ;
# Bus = (check, top).(reset, r2).Bus ;

% reward rate-trans = /\{trans} & [trans,0.01]\/{trans} => rate(trans) ;

Dev <L> Bus
```

[The PEPA Workbench generates:]

```
rate-trans := 0
+P[3] * data  % Con <L> (reset, r2).Bus
+P[6] * data  % Con <L> Bus
;
```

Figure 3.3: Using the Reward language in the PEPA Workbench

formula, the algorithm begins with the set of derivatives of P; then restricts to those which were reached by activities of type α; then further restricts to those which recursively satisfy formula G. The rate from P to this subset is then generated, and compared against μ.

The algorithm as it stands is a naïve implementation in order to demonstrate proof of principle. For example speed increases may be possible by checking the satisfaction of all subformulas of a formula F while working with any given derivative.

```
fun modelCheck (And(f1,f2)) P = (modelCheck f1 P) andalso
                                (modelCheck f2 P)
  | modelCheck (Mtt) P = true
  | modelCheck (Dia(a,mu,G)) P =
    let
       val dP = derivatives P
       val dPa = chooseByType a dP
       val dPf = chooseBySat G dPa
    in
       not (lessThan (sumRates dPf, mu))
    end
  | modelCheck ...
```

Figure 3.4: Excerpt of the PML_{μ} model checking algorithm

3.6 Summary

To summarise, this chapter presents a language which can be used to automatically generate a reward structure over the stochastic process of a PEPA model. The language is based on a modal logic, PML_μ and allows the modeller to 'focus' on model subcomponents, by the use of contexts. It is shown that PML_μ characterises PEPA's strong equivalence. Moreover, conditions are given describing which parts of a PEPA model may be aggregated without affecting the reward structure obtained. Finally it is conjectured that these aggregation conditions can be made more liberal if a restricted form of PML_μ formula is employed.

There is scope to improve upon these results. For example, the current definitions do not allow a subcomponent to be aggregated if it contains a hiding operator. This restriction ensures that if any action type can be witnessed at a contextual level, then it may always be witnessed. This is unnecessarily strict, since the modeller may wish to study the ability of a model to perform an activity in context which is completely unaffected by a particular hiding operator. Refining this definition would increase the usefulness of the PEPA Reward language. A criticism which has been aimed at the current approach is that the modal operator specifies rate-based properties of a PEPA process for a given action type, but does not take into account the *total* rate of exit from a given state of the process. The motivation is that a modeller specifying properties of a PEPA process may be misled by a negative result on the departure rate from a state via a particular action type, when in fact the total exit rate from a given state is sufficient for the required property. This opinion seems to depend on the application of the modal logic. It is shown in Section 3.4 that the current intepretation may be used successfully for calculating simple rate-based performance measures. Whatever becomes the prevailing view, Conjecture 1 hints that *rate-free* formulae can be useful for specifying types of rewards, and may also lead to more general aggregations while still preserving specified performance measures.

Chapter 4

A Stochastic Process Algebra Structure for Insensitivity

4.1 Introduction

The following two chapters use insensitivity results to examine cases where generally distributed activities may be introduced into PEPA models. This chapter gives details of a derived algebraic combinator which guarantees insensitivity for some model activities. The following chapter provides a general study, in terms of balance equations only.

The exponential assumption inherent in Markovian analysis is regarded by some as a restriction in the application of SPA modelling. For example, deterministic random variables may be more appropriate in modelling time-outs in communication protocols. It would therefore be of great utility to be able to incorporate more general distributions into SPA performance models. Indeed several attempts to do this are underway, as discussed in Section 2.5.2. However, unlike previously published work, the aim of this chapter is to introduce this increased modelling expressiveness only when it does not seriously impinge on model tractability. To justify the approach, the concept of insensitivity is used. A stochastic process is said to be insensitive if its steady-state distribution depends on the distribution of one or more of its state lifetime random variables *only through the mean.* In this thesis, insensitivity is considered with respect to the activities of an SPA model when possible. The consequence of an SPA model being insensitive to a particular activity is that the activity may provably be arbitrarily distributed (with the same mean) without affecting the steady-state probability distribution of the model. Therefore conditions under which activities can be shown to be insensitive are conditions under which more flexible modelling features, such as realistic time-outs and repair times, may be introduced.

This chapter studies a structure of SPA model from which it is possible to infer the insensitivity of particular activities. This class of PEPA models may be constructed using a new derived combinator, and consists of a collection of simple subcomponents which interact in a weak fashion. Despite this interaction, it is shown that insensitivity of residence time in particular sub-states (states of each subcomponent) is retained, and thus in this approach, insensitivity of particular SPA activities too. Therefore, in many cases, generally distributed activities may be used to build the subcomponents without affecting the equilibrium distribution of the model as a whole. Models built using this derived combinator are guaranteed to have the insensitivity properties defined in this chapter. Models which are insensitive to some of their distributions but not to others are not considered in this chapter.

In Section 4.2 insensitivity is introduced, and the basic results from the literature are presented. Section 4.3 describes the structure of PEPA models which then exhibit the insensitivity property, and this class is demonstrated with a simple example in Section 4.3.4. Since it is found that in fact the solutions to models in this class exhibit product form, Section 4.4 looks at existing product form solutions for process algebra, and compares them to this new approach.

4.2 Insensitivity

This section begins with an explanation of insensitivity, and the stochastic model which is used as a vehicle for the insensitivity property is formally introduced. The conditions on the stochastic process that guarantee insensitivity are then described, leading to a discussion of insensitivity in stochastic Petri net models.

A stochastic process is said to be insensitive if its steady-state distribution depends on the distribution of one or more of the random variables representing residence time in a state *only through their mean.* Intuitively, this means that a random variable may be replaced with another with an identical mean, while preserving the steady-state solution of the process. Just as steady-state is characterised by a set of *global balance equations*, insensitivity of residence time in a state is characterised by a set of *insensitivity balance equations.* These are interpreted with respect to a particular model, which is introduced next.

4.2.1 The Generalised Semi-Markov Process

A Generalised Semi-Markov Process (GSMP) is defined on a set of states $\{g \mid g \in G\}$. Within each of these states are active elements, $s \in S$. The current state g will contain a set of active elements, each element s with a *lifetime* which decays at the state dependent rate $c(s, g)$. Let S', S^* be disjoint, and such that $S' \cup S^* = S$. If $s \in S'$ then the lifetime of s is exponentially distributed; if $s \in S^*$, s has a generally distributed lifetime. When the lifetime of an active element s expires, the process moves to another state $g' \in G$ with probability $p(g'; g, s)$. When the process changes from state g due to the death of an active element the remaining elements from $g \cap S^*$ retain their spent lifetimes. Active elements new to the current state are given new lifetimes, as samples drawn from their governing distribution functions. A restriction on the process's behaviour is that no two active elements from S^* may be activated or die simultaneously. An excellent operational view of the execution of a GSMP is given by Shedler [69].

Insensitivity was first studied as an end in itself in the early 1960s. Results were originally presented with respect to the Generalised Semi-Markov Scheme, which generates a stochastic process which is a GSMP. Working with this model, Matthes [58] showed:

Theorem 4.2.1. (Matthes) *The following two statements are equivalent:*

1. *The process is insensitive to the elements of S^*. That is, the distributions of the lifetimes of the elements of S^* may be replaced by any other distribution with the same mean, while still retaining the same equilibrium distribution.*
2. *When all elements of S^* are assumed to be exponentially distributed, the flux out of each state due to the death of an element of S^* is equivalent to the flux into that state due to the birth of that element.*

The second statement describes the *insensitivity balance equations* for the GSMP. A GSMP state contains a set of active elements, and the state may be left due to the death of any of them. The flux out of a state *due to the death of an element* contributes to the total flux out of a state, and is given by the probability of being in that state multiplied by the rate at which that state is left due to the fact that the lifetime of a given active element has expired (cf. Section 2.2.4.1). The *birth of an element* refers to the point at which an active element becomes enabled and is given a new lifetime. The flux into a state *due to the birth of an element* is given by, for all predecessor states, the probability of being in such a state multiplied by the rate of departure into the current state such that the active element is given a new lifetime in the current state. Notice that the theorem gives conditions for insensitivity of *all* generally distributed active elements. In this chapter, the only models considered are those which are insensitive to all their generally distributed active elements.

4.2.2 Conditions for Insensitivity

High-level performance modelling paradigms provide features which are 'active', in some sense, for a randomly distributed length of time. For example, SPNs have transitions, and SPAs typically provide activities. It is these features for which it would be useful to relax the exponential assumption. However exploiting insensitivity is not straightforward, and this is because a feature which appears once in a high-level model may be represented in many states of the underlying stochastic process.

Henderson and Lucic [37] consider the conditions under which a GSMP may instead be produced from an SPN. In their translation, SPN transitions are represented by active elements; in this setting, all concurrently enabled generally distributed transitions carry over their spent lifetimes to successor markings. The authors give a translation of Matthes' theorem for use with stochastic Petri nets:

Corollary 2 (Henderson and Lucic). *The following two statements are equivalent:*

1. *The SPN model is insensitive to each generally distributed transition* t*.*
2. *The purely Markov process, i.e. when* $S = S'$*, has the property that for all markings* j *that enable transition* t*, the flux into* j *enabling* t *is balanced by the flux out of* j *due to the death of* t*.*

In general, the choice of successor state is time-dependent; consider two enabled transitions, one of which, t, is uniformly distributed between m and n. If n time units elapse, then t must definitely have fired. If a transition fires before m time units pass, it must not have been t. This property is called *age dependent routing*. Fortunately, a result due to Rumsiewicz and Henderson [65] states that the *time-averaged* stochastic process, with age independent transition probabilities, gives the same equilibrium distribution as the original process, under the condition that the time-averaged process is insensitive to its generally distributed transitions. The technical difficulty is constructing the time-averaged mean sojourn time, given a set of transitions with arbitrary distributions, and then the next state probabilities. Although our approach will focus on insensitivity of SPA activities, it also encounters this difficulty. Matthes theorem is used in an SPA setting in Section 4.3.2, where a GSMP semantics is given to a particular structure of PEPA models; this makes precise what is meant by the insensitivity of PEPA activities.

4.3 A Structure for Generally Distributed Concurrently Enabled Non-Conflicting Activities

In this section, a derived combinator is presented which allows the construction of models containing concurrently enabled activities and which are insensitive. Therefore, these activities may be generally distributed. Sequential components directly generate an SMP and are insensitive to all activities; however the models constructed here are not sequential components in general. In essence, the models consist of a set of concurrent sequential components, with a synchronisation discipline. This takes the form of an arbiter process which synchronises with two or more sequential components. The synchronisation enforces a *queueing* discipline and causes each sequential component to wait in a queue at a particular point in its lifecycle. When a process leaves a particular queue, it does so with a fixed rate particular to that queue. However, despite such interaction, all activities not enabled while in the queue are insensitive to their distributions.

In this section, the notion of the insensitivity of a PEPA activity is made precise. Section 4.3.1 proceeds to define a new combinator, $Q_{A,\xi}(\cdot)$, leading to two results. Theorem 4.3.1 demonstrates the general solution form of queueing discipline models; and Theorem 4.3.2 proves the insensitivity of a particular set of activities used in queueing discipline models.

4.3.1 A Derived Combinator for the Queueing Discipline

In this section, the new combinator, $Q_{A,\xi}(\cdot)$, is introduced. It can be used to build PEPA models insensitive to the distributions associated with particular activities. Assume an arbitrary PEPA model $\prod_{i=1}^{n} S_i$, therefore with no cooperations, upon which a queueing discipline is to be enforced. Now a derived combinator is given, which defines a cooperating process R_A and a cooperation set M_A, dependent on a set of action types A.

Definition 4.3.1 (Simple Queueing Combinator).

$$Q_A(S_1, \ldots, S_n) \stackrel{\text{def}}{=} (\prod_{i=1}^{n} S_i) \underset{M_A}{\bowtie} R_A \tag{4.3.1}$$

An intuitive meaning for this notation is that $Q_A(S_1, \ldots, S_n)$ allows each $S_i, 1 \leq i \leq n$, to proceed in parallel, independent of each other, and only enforces synchronisation of each S_i with a distinguished process, on action types determined by each S_i and the set A. If each S_i of a subset of processes currently enables an activity whose action type is in A, each S_i must 'queue' to perform its activity. Therefore, at any one time, only one of the queueing S_i processes is allowed to proceed.

Let A be such that for each $\alpha \in A$, there exists a unique S_i such that $(\alpha, r) \in \overrightarrow{\mathcal{A}ct}(S_i)$ for some r; and for each S_i, there exists a unique $\alpha \in A$ such that $(\alpha, r) \in \overrightarrow{\mathcal{A}ct}(S_i)$ for some r. Next some definitions are given which allow the specification of a cooperation set M_A called the *arbiter synchronisation set*. This procedure is mechanical, and could be simply automated.

Definition 4.3.2 (Enabling action type). *Let P be a process, and* $\mathtt{a}$ *an activity. Then β is an* enabling action type for $\mathtt{a}$ *in P if*

- *for all derivatives P' of P such that $P' \xrightarrow{\mathtt{a}}$, there exists a derivative $P'' \not\equiv P'$ such that $P'' \xrightarrow{(\beta,r)} P'$ for some r,*
- *for every derivative P'' of P such that for some r, $P'' \xrightarrow{(\beta,r)} P'$, it is the case that $P' \xrightarrow{\mathtt{a}}$.*

An enabling action type for $\mathtt{a}$ can be viewed as the type of an activity which may only be performed immediately prior to the model enabling $\mathtt{a}$, and which if performed, always leads to $\mathtt{a}$ being enabled. The set $e_A(P)$ is a set of enabling action types for those activities of P with types present in A. An assumption is made that for i, j, the enabling action types of process S_i are distinct from those of S_j, that is that $e_A(S_i) \cap e_A(S_j) = \emptyset$, for $i \neq j$. This ensures there is no confusion over which component is about to enter a particular queue. Furthermore, for the models studied in this chapter, it is assumed that for every queue activity $\mathtt{a}$ that may be enabled by a process P, there exists an enabling action type for $\mathtt{a}$ in P.

Definition 4.3.3 (Arbiter synchronisation set). *Let $P \equiv Q_A(S_1, \ldots, S_n)$ be a process with a queueing discipline. The* arbiter synchronisation set *M_A of P is given by $A \cup e_A(P)$.*

From here, a process R_A is defined which enforces the required queueing discipline. R_A is called an *arbiter process*.

Definition 4.3.4 (Arbiter process). *Let $P \equiv Q_A(S_1, \ldots, S_n)$ be a process with a queueing discipline. Let $\overrightarrow{\mathcal{A}ct}_A(P) \stackrel{\text{def}}{=} \{(\alpha, r) \in \overrightarrow{\mathcal{A}ct}(P) : \alpha \in A\}$ denote the set of queue activities belonging to process P.*

The arbiter process *for P is given by $R_{A,\langle\rangle}$, defined as*

$$
\begin{array}{rcl}
R_{A,\langle\rangle} & \stackrel{\text{def}}{=} & \sum_{1\leq i\leq n} \sum_{\alpha\in A} \sum_{\beta\in e_\alpha(S_i)} (\beta,\top).R_{A,\langle S_i\rangle} \\
R_{A,\langle S_j,S_m,\dots,S_n\rangle} & \stackrel{\text{def}}{=} & \sum_{\substack{1\leq i\leq n \\ i\notin\{j,m,\dots,n\}}} \sum_{\alpha\in A} \sum_{\beta\in e_\alpha(S_i)} (\beta,\top).R_{A,\langle S_j,S_m,\dots,S_n,S_i\rangle} \\
 & + & \sum_{(\alpha,r)\in \overrightarrow{Act}_A(S_j)} (\alpha,\top).R_{A,\langle S_m,\dots,S_n\rangle}, \quad |\{S_j,S_m,\dots,S_n\}| < n \\
R_{A,\langle S_j,S_m,\dots,S_n\rangle} & \stackrel{\text{def}}{=} & \sum_{(\alpha,r)\in \overrightarrow{Act}_A(S_j)} (\alpha,\top).R_{A,\langle S_m,\dots,S_n\rangle}, \quad |\{S_j,S_m,\dots,S_n\}| = n
\end{array}
$$

This arbiter process can be viewed as a PEPA definition of a queue, which enforces an ordering on the processes it controls. The size of the queue specified by the combinator is equal to the number of processes given as the combinator's arguments.

4.3.1.1 A Restriction on Activity Rates

The arbiter process given in Definition 4.3.4 only performs activities which are passive with respect to the model with which it interacts. This simplifies the definition, and ensures that the arbiter does not affect the *rate* at which any activities are performed. However, in order to gain insensitivity results, an extra restriction is required:

> All queued processes *must* perform their queue activities with a rate fixed for the particular arbiter process and number of customers in the queue (that is, the mean of the distribution with which a queued process performs its queue activity is common to all queued processes, and may vary only with queue length).

It is a mechanical task to alter a PEPA model with a queueing discipline such that it conforms to this restriction. Recall that the PEPA definition of a model with a queueing discipline over the action set A is given by

$$
Q_A(S_1,\dots,S_n) \stackrel{\text{def}}{=} (\prod_{i=1}^{n} S_i) \underset{M_A}{\bowtie} R_A
$$

where R_A is an arbiter process and M_A is an arbiter synchronisation set. Now a simple translation of the current queueing discipline model is defined such that it conforms with the required rate restriction. Crucial to this translation is the use of PEPA's passive activities. Currently, the arbiter restricts the behaviour of processes, but does not affect the rate at which they perform activities. The idea behind the translation is that each process's right to individual behaviour in the queue is removed, by making the queue activity passive, so that the queue defines at what rate processes may pass through.

Definition 4.3.5 (Rate replacement). *Let P be a PEPA process and A a set of action types. Then $P_{A\longrightarrow\top}$ is the PEPA process where each occurrence of a non-passive activity (α,r) such that $\alpha\in A$ is modified such that it becomes passive, that is it is changed to $(\alpha,\top)$. $P_{A\longrightarrow\top}$ is defined on the structure of P as*

$$\begin{array}{rcl}
P \equiv Q \underset{L}{\bowtie} R & : & Q_{A\longrightarrow\top} \underset{L}{\bowtie} R_{A\longrightarrow\top} \\
P \equiv Q + R & : & Q_{A\longrightarrow\top} + R_{A\longrightarrow\top} \\
P \equiv Q/L & : & Q_{A\longrightarrow\top}/L \\
P \equiv (\alpha, r).Q & : & (\alpha, \top).Q_{A\longrightarrow\top} \ \textit{if } \alpha \in A \\
P \equiv (\alpha, r).Q & : & (\alpha, r).Q_{A\longrightarrow\top} \ \textit{if } \alpha \notin A \\
\textit{otherwise} & : & Q_{A\longrightarrow\top} \ \textit{where } P \stackrel{\text{def}}{=} Q
\end{array}$$

Now, given a queueing discipline model $Q_A(S_1, \ldots, S_n)$, let r_i represent the rate at which S_i performs its queue activity. The following modification to the arbiter process is made.

Definition 4.3.6 (Rate restricted arbiter process). *Let $P \equiv Q_A(S_1, \ldots, S_n)$ be a process with a queueing discipline. Let ξ be a sequence of rates $\langle \xi_1, \ldots, \xi_n \rangle$, which defines the rate at which any S_i will perform its queue activity dependent on the current queue length. Then the arbiter process* rate restricted by ξ, *R^{ξ}_A, is defined as*

$$\begin{array}{rcl}
R^{\xi}_{A,\langle\rangle} & \stackrel{\text{def}}{=} & \sum_{1\le i\le n} \sum_{\alpha\in A} \sum_{\beta\in e_\alpha(S_i)} (\beta, \top).R^{\xi}_{A,\langle S_i\rangle} \\
R^{\xi}_{A,\langle S_j, S_m, \ldots, S_n\rangle} & \stackrel{\text{def}}{=} & \sum_{\substack{1\le i\le n \\ i\notin\{j,m,\ldots,n\}}} \sum_{\alpha\in A} \sum_{\beta\in e_\alpha(S_i)} (\beta, \top).R^{\xi}_{A,\langle S_j, S_m, \ldots, S_n, S_i\rangle} \\
& + & \sum_{(\alpha,r)\in\overrightarrow{Act}_A(S_j)} (\alpha, \xi_j).R^{\xi}_{A,\langle S_m,\ldots,S_n\rangle}, \qquad |\langle S_j, S_m, \ldots, S_n\rangle| < n \\
R^{\xi}_{A,\langle S_j, S_m, \ldots, S_n\rangle} & \stackrel{\text{def}}{=} & \sum_{(\alpha,r)\in\overrightarrow{Act}_A(S_j)} (\alpha, \xi_j).R^{\xi}_{A,\langle S_m,\ldots,S_n\rangle}, \qquad |\langle S_j, S_m, \ldots, S_n\rangle| = n
\end{array}$$

Now it is a simple task to write out the definition of the rate restricted queueing discipline model.

$$Q_{A,\xi}(S_1, \ldots, S_n) \stackrel{\text{def}}{=} (\prod_{i=1}^{n} S_{iA\longrightarrow\top}) \underset{M_A}{\bowtie} R^{\xi}_{A,\langle\rangle} \tag{4.3.2}$$

4.3.1.2 Multiple Queues

Now a definition is given for a model with *several* queueing disciplines. This is a straightforward extension of the model structure used to build models with one queueing discipline. For $1 \le i \le N$, let A_i be a set of action types such that for $1 \le i < j \le N$, $A_i \cap A_j = \emptyset$; $A_* = \cup_{i=1}^{N} A_i$; and ξ_i a finite sequence of rates, as before.

Definition 4.3.7 (Extended Queueing Combinator).

$$Q_{\langle A_1,\xi_1,\ldots,A_N,\xi_N\rangle}(S_1, \ldots, S_n) \stackrel{\text{def}}{=} (\ldots((\prod_{i=1}^{n} S_{iA_*\longrightarrow\top}) \underset{M_{A_1}}{\bowtie} R^{\xi_1}_{A_1}) \ldots \underset{M_{A_N}}{\bowtie} R^{\xi_N}_{A_N}) \tag{4.3.3}$$

This notation is cumbersome, and so the shorthand $Q_\chi(S_1, \ldots, S_n)$ is used to represent $Q_{\langle A_1,\xi_1,\ldots,A_N,\xi_N\rangle}(S_1, \ldots, S_n)$.

Now, an important property of these models is exhibited with a simple lemma.

Lemma 4.3.1. $Q_\chi(S_i) \xrightarrow{(\alpha,r)} Q_\chi(S_i')$ *if and only if*

$$Q_\chi(S_1,\ldots,S_i,\ldots,S_n) \xrightarrow{(\alpha,r)} Q_\chi(S_1,\ldots,S_i',\ldots,S_n)$$

Proof. Both directions are proven.

Case $\Longrightarrow$: By Definition 4.3.7, $Q_\chi(S_i)$ is of the form

$$(\ldots((S_{iA_*\longrightarrow\top} \underset{M_{A_1}}{\bowtie} R_{A_1}) \underset{M_{A_2}}{\bowtie} R_{A_2})\ldots \underset{M_{A_N}}{\bowtie} R_{A_N})$$

for particular R_{A_j} and M_{A_j}. Consider any inference tree for $Q_\chi(S_i) \xrightarrow{\mathtt{a}} Q_\chi(S_i')$. Since each R_{A_j} is restricted such that it can only evolve in cooperation, then necessarily any $\mathtt{a}$-transition made by $Q_\chi(S_i)$ must derive from a transition of the form $S_{iA_*\longrightarrow\top} \xrightarrow{(\alpha,s)} S'_{iA_*\longrightarrow\top}$ for some s. Now by the operational semantics of PEPA, it is possible to infer

$$\prod_{j=1}^{N} S_{jA_*\longrightarrow\top} \xrightarrow{(\alpha,s)} S_{1A_*\longrightarrow\top} \mid\mid \ldots \mid\mid S'_{iA_*\longrightarrow\top} \mid\mid \ldots \mid\mid S_{NA_*\longrightarrow\top} \text{ for some } 1 \leq i \leq N$$

and therefore that

$$Q_\chi(S_1,\ldots,S_i,\ldots,S_n) \xrightarrow{(\alpha,r)} Q_\chi(S_1,\ldots,S_i',\ldots,S_n) \text{ for some rate } r$$

Case $\Longleftarrow$: By a reverse argument, any inference tree leading to

$$Q_\chi(S_1,\ldots,S_i,\ldots,S_n) \xrightarrow{(\alpha,r)} Q_\chi(S_1,\ldots,S_i',\ldots,S_n)$$

can also be used to infer

$$\prod_{j=1}^{N} S_{jA_*\longrightarrow\top} \xrightarrow{(\alpha,s)} S_{1A_*\longrightarrow\top} \mid\mid \ldots \mid\mid S'_{iA_*\longrightarrow\top} \mid\mid \ldots \mid\mid S_{NA_*\longrightarrow\top}$$

and since every cooperation set is empty, this can lead to

$$\begin{aligned} &(\ldots((S_{iA_*\longrightarrow\top} \underset{M_{A_1}}{\bowtie} R_{A_1}) \underset{M_{A_2}}{\bowtie} R_{A_2})\ldots \underset{M_{A_N}}{\bowtie} R_{A_N}) \\ &\xrightarrow{(\alpha,r)} (\ldots((S'_{iA_*\longrightarrow\top} \underset{M_{A_1}}{\bowtie} R_{A_1}) \underset{M_{A_2}}{\bowtie} R_{A_2})\ldots \underset{M_{A_N}}{\bowtie} R_{A_N}) \end{aligned} \tag{4.3.4}$$

and thus $Q_\chi(S_i) \xrightarrow{(\alpha,r)} Q_\chi(S_i')$.

□

4.3.2 Mapping a PEPA Queueing Discipline Model to a GSMP

To study the insensitivity of activities in queueing discipline models, this section provides a translation to a GSMP. Such a mapping was given by Hillston [43] for earlier work on examining insensitivity of PEPA models. In this setting, a mapping is given for a PEPA model as a configuration of *top-level* components, for example for a cooperation of two sequential components. The active elements of the GSMP are multisets of enabled activities. Such a cooperation which would in general enable three active elements—one for the multiset of individual activities that each component could perform individually, and one for the activities on which each component must cooperate. For example, consider the PEPA component

$$((\alpha, r_1).P_1 + (\gamma, r_2).P_2) \underset{\{\alpha,\beta\}}{\bowtie} ((\alpha, s_1).Q_1 + (\beta, s_2).Q_2 + (\gamma, s_3).Q_3) \tag{4.3.5}$$

Under Hillston's mapping, this component is represented by the GSMP state

$$\begin{gathered}(\{(\gamma, r_2)\}, \{(\gamma, s_3)\}, \{(\alpha, t_1)\}) \\ \text{where } t_1 = \min(r_1, s_1)\end{gathered} \tag{4.3.6}$$

The active elements present represent the individual abilities of both sides of the cooperation to proceed, and one element to represent the shared ability of the component to proceed via cooperation. The PEPA model structure studied in this chapter is of a restricted form, and so the GSMP mapping provided is more limited, and in particular does not need to take account of arbitrary cooperation in PEPA models.

The approach used in this chapter is based on the consideration of PEPA models of the form described in Section 4.3.1.2, that is, a collection of concurrent sequential components with queueing disciplines. In this approach, an active element of the GSMP is constructed from a sequential component as a multiset of pairs, where each pair is an enabled activity and the PEPA derivative that results. Without a queueing discipline, only simple models need be considered; these consist of the unrestricted cooperation of sequential PEPA components. For example, consider the component given below.

$$((\alpha, r_1).P_1 + (\gamma, r_2).P_2) \;||\; (\beta, s_1).Q_1 \tag{4.3.7}$$

Under the new mapping, this component would be represented by the GSMP state

$$(\{\!|((\alpha, r_1), P_1), ((\gamma, r_2), P_2)|\!\}, \{\!|((\beta, s_1), Q_1)|\!\}) \tag{4.3.8}$$

The queueing discipline consists of a component, the arbiter process, with the potential to interact with each of the sequential components; however the component has no individual ability, and must cooperate in order to evolve. In this respect, the queueing discipline provides a similar restriction to that described by Hillston in [46]. Its effect on the GSMP state is not to add active elements, but rather to reduce the number of activities which constitute existing active elements.

Now a more formal definition is presented. Assume a PEPA queueing discipline model $P \equiv Q_\chi(S_1, \ldots, S_n)$. Each state $g \in G$ is such that $g \in \prod_{i=1}^{n} S_i$, that is each state consists of a size n cartesian product of active elements. For the form of model considered here, there is a one-to-one mapping between GSMP states and derivatives $P' \in ds(P)$.

Definition 4.3.8 (GSMP state). *The GSMP state for derivative* $P \equiv Q_\chi(S_1, \ldots, S_n)$ *is given by*

$$\prod_{i=1}^{n} \{\!|(\mathtt{a}, Q_\chi(S_i')) : Q_\chi(S_i) \xrightarrow{\mathtt{a}} Q_\chi(S_i')|\!\} \tag{4.3.9}$$

(Note that this is a conventional cartesian product over sets.) The GSMP state representation of a component P is denoted $\hat{P}$. This definition gives a GSMP with some simple properties. The completion of an activity results in the death of an active element; Lemma 4.3.1 shows that with the particular restrictions that our queueing discipline places on the sequential components that the death of an active element corresponding to S_i for some i will not cause the death of any active elements for components S_j, $j \neq i$. Since interest will be limited to studying when activities, and thus active elements, can be generally distributed, this property is important—it ensures that the remaining lifetime of an active element representing an activity will persist over a state change, unless the state change corresponds to the completion or disabling of that activity. Since this is an alternative model for a PEPA process, it is important that it preserves the performance properties of the original model too. The lifetime of an active element is defined next.

Definition 4.3.9 (Active Element Lifetime). *Let s be an active element present in state $\hat{P}$. The lifetime of s is exponentially distributed with mean*

$$\Bigg(\sum_{\{\!|((\alpha, r), Q_\chi(S')) \in s|\!\}} r \Bigg)^{-1} \tag{4.3.10}$$

The instantaneous rate of departure from state $\hat{P}$ is denoted by $\hat{q}(\hat{P})$, and given by $\sum_{i=1}^{n} \sum_{(\alpha, r) \in \overrightarrow{Act}(Q_\chi(S_i))} r$.

Since an active element is a *multiset* of activities, then the death of an active element also corresponds to the disabling of more than one activity in general. This interpretation is correct, since the multiset will represent *competing*, and therefore mutually disabling, activities, as the result of a sequential component offering a choice. When an active element s of a state $\hat{P}$ dies, the probability that the next state is $\hat{P'}$ is given by a probability distribution $p(\hat{P'}; \hat{P}, s)$. This is defined as follows.

Definition 4.3.10 (GSMP next-state probabilities). *Let* $P \equiv Q_\chi(S_1, \ldots, S_n)$, *and* $P' \equiv Q_\chi(S_1, \ldots, S_j', \ldots, S_n)$ *such that* $P \longrightarrow P'$. *Given that* s_i *is the active element that dies in state* $\hat{P}$, *the probability that the next state is* $\hat{P}'$ *is given by*

$$p(\hat{P}'; \hat{P}, s_i) = \mathbf{1}(j = i)\Big(\sum_W r / \sum_V r\Big) \tag{4.3.11}$$

where $W = \{\!|((\alpha, r), Q_\chi(S_i')) \in s_i : Q_\chi(S_i) \xrightarrow{(\alpha,r)} Q_\chi(S_i')|\!\}$ *and* $V = \{\!|((\alpha, r), Q_\chi(S_i')) \in s_i|\!\}$. *The probability that the next state is* $\hat{P}'$ *is denoted by* $\hat{p}(\hat{P}, \hat{P}')$, *and is given by* $\sum_{i=1}^n p(\hat{P}'; \hat{P}, s_i)$.

Although not concise, these definitions are intuitive. An active element s of a state $\hat{P}$ is a multiset of pairs of activities and derivatives. The activities constitute a particular subset of the enabled activites of P. These are used to specify the mean lifetime of s. The derivatives capture how the GSMP can evolve on the death of s, and are used to define the successor state probability distribution.

Now some simple results about the GSMP model are proved. First it is shown that the GSMP model is a faithful representation of the original PEPA model; in particular that performance properties are preserved.

Lemma 4.3.2. *Let* $P \equiv Q_\chi(S_1, \ldots, S_n)$. *Then* $q(P) = \hat{q}(\hat{P})$.

Proof. By definition (see [44]), $q(P) = \sum_{(\alpha,r)\in\overrightarrow{\mathcal{A}ct}(P)} r$. Now,

$$\begin{aligned}
\hat{q}(\hat{P}) &= \sum_{i=1}^{n} \sum_{(\alpha,r)\in\overrightarrow{\mathcal{A}ct}(Q_\chi(S_i))} r && \text{(by Definition 4.3.9)}\\
&= \sum_{(\alpha,r)\in\overrightarrow{\mathcal{A}ct}(Q_\chi(S_1,\ldots,S_n))} r && \text{(by Lemma 4.3.1)}\\
&= \sum_{(\alpha,r)\in\overrightarrow{\mathcal{A}ct}(P)} r\\
&= q(P)
\end{aligned}$$

$\square$

Lemma 4.3.3. *Let* $P \equiv Q_\chi(S_1, \ldots, S_n)$, *and* $P \xrightarrow{(\alpha,r)} P'$ *(therefore* $P' \equiv Q_\chi(S_1, \ldots, S_j', \ldots, S_n)$ *for some* j*). Then* $p(P, P') = \hat{p}(\hat{P}, \hat{P}')$.

Proof.

$$\begin{aligned}
\hat{p}(\hat{P},\hat{P}') &= \sum_{i=1}^{n} p(\hat{P}';\hat{P},s_i)\cdot \Pr(s_i \text{ dies first}) \\
&= \sum_{i=1}^{n} \mathbf{1}(j=i)\Big(\sum_{W} r/\sum_{V} r\Big)\cdot\Big(\sum_{V} r/\sum_{(\alpha,r)\in\overrightarrow{\mathcal{A}ct}(P)} r\Big) \quad \text{(by Definition 4.3.10)} \\
&= \sum_{W} r/\sum_{(\alpha,r)\in\overrightarrow{\mathcal{A}ct}(P)} r \\
&= \sum_{U} r/\sum_{(\alpha,r)\in\overrightarrow{\mathcal{A}ct}(P)} r \\
&= p(P,P')
\end{aligned}$$

where

$$\begin{aligned}
U &= \{\!|(\alpha,r): Q_{\chi}(S_1,\ldots,S_j,\ldots,S_n)\xrightarrow{(\alpha,r)} Q_{\chi}(S_1,\ldots,S_j',\ldots,S_n)|\!\} \\
W &= \{\!|((\alpha,r),Q_{\chi'}(S_i'))\in e_i : Q_{\chi}(S_i)\xrightarrow{(\alpha,r)} Q_{\chi'}(S_i')|\!\} \\
V &= \{\!|((\alpha,r),Q_{\chi}(S_i'))\in e_i|\!\}
\end{aligned}$$

□

These results are almost self-evident by deliberate construction of the GSMP model, but they illustrate that the GSMP performance model will faithfully represent the PEPA model.

For a PEPA model, this GSMP mapping would result in active elements with exponential lifetimes. However, in particular circumstances, generally distributed active elements will be allowed, and it is shown that these do not affect the steady-state solution. At the process algebra level, the modeller would wish to use general distributions when describing the duration of an activity. A generally distributed active element representing a set of (albeit mutually disabling) activities results in a next state probability which is well-defined, but which is in general difficult to calculate. However, note that where a sequential component does not currently enable a choice, then the active element is represented by a single activity. In such circumstances, the next state probability is trivial. Furthermore, despite the loss of behavioural independence of each sequential component due to the restrictions of the queueing discipline, it is still the case that such activities are provably insensitive. With this mapping, an analogous form of Matthes' theorem (4.2.1) can be used for PEPA models.

Corollary 3. *The following two statements are equivalent:*

1. *The PEPA model is insensitive to each generally distributed activity* `a`.

2. *The purely Markov process, i.e. when $S = S'$, has the property that for all states P that enable activity* a*, the flux into P enabling* a *is balanced by the flux out of P due to the completion or disabling of* a.

Comparing this with Corollary 2, it can be seen that the condition for the flux out of a state looks to be different. The explanation for this is that each state of the GSMP for a queueing discipline model is constructed as a multiset of pairs of activities and PEPA terms, and the theory of insensitivity deals with the death of active elements of the GSMP. If an active element of a queueing discipline model 'dies', one of the activities that makes up the active element completes, and all the others are disabled.

4.3.3 Queueing Discipline Structure and Insensitivity

Given the process algebra definitions of the queueing discipline framework above, it is now possible to formally analyse the structure of the models it produces. The aim of the work in this chapter is to examine conditions for insensitivity of activities in process algebra models. The first result gives insight into why the structure of models built with the new combinator is related to insensitivity; it is shown that the solution of such models is a product form over both the solutions to the individual sequential components, and the current queue lengths.

Let there be a size-n set of sequential PEPA components S_i, that is a set of components each without the syntactic cooperation or hiding operators. Recall from Chapter 2 that if P is a PEPA process, then ${}^{\diamond}P$ is a process such that ${}^{\diamond}P \longrightarrow P$.

Now assume a model of the form $Q_{\chi}(S_1, \ldots, S_n)$ where $\chi = \langle A_1, \xi_1, \ldots, A_N, \xi_N \rangle$. Therefore, A_j defines the set of activities by which components S_i may leave the jth queue, and also implicitly defines which components S_i may enter the queue in the first place. It is assumed that for any $0 \leq i < j \leq N$, the sets of enabling action types e_{A_i} and e_{A_j} are pairwise disjoint; this ensures that at each point in the evolution of each S_i, it may enter either one queue only, or none at all. Furthermore, it is assumed that if a sequential process, S_i, enables a queue activity, then it has no potential to perform another activity (effectively, it must wait its turn in the queue); and that for every queue activity of S_i, there exists an enabling action type for that activity in S_i. Currently, processes may not leave a queue such that they move directly into another queue, that is for any $0 \leq i < j \leq N$, $e_{A_i} \cap A_j = \emptyset$. This restriction is only in place for simplicity, and indeed it is conjectured that the restriction is unnecessary. However the extension remains as work to be done.

At any point, a number of processes may be queued due to the jth queue. Let q_j represent the current queue length and d_j the maximum length (i.e. $d_j = |\xi_j|$), such that $0 \leq q_j \leq d_j$. Let Θ represent the set of processes currently queueing; Θ_j those currently queueing in the jth queue. If S_i is currently at the front of the jth queue, it may perform its queue activity $\mathtt{a}$ with action type in A_j at a rate ξ_{jq_j}. Further, H_j denotes the index of the process at the head of the jth queue, and T_j denotes the index of the process at the tail. If the jth queue is empty, both of these variables are undefined. The notation is with respect to the state currently being considered, where there is no ambiguity. For convenience, ξ_{j0} is defined to be 0, for each j.

Finally, each sequential component is a PEPA process in its own right, and as such, its underlying stochastic process has a steady-state probability distribution. Let $\pi_i(\cdot)$ denote the steady-state probability distribution of S_i; then if $S_i{}' \in ds(S_i)$, the long-run probability of being present in $S_i{}'$ is given by $\pi_i(S_i{}')$. With all notation in place, the theorem can now be stated:

Theorem 4.3.1. *Let $P \equiv Q_\chi(S_1, \ldots, S_n)$. The steady-state solution of P is given by*

$$\pi(P) = \frac{1}{G}\prod_{i=1}^{n}\pi_i(S_i) \cdot \prod_{i=1}^{N}\prod_{j=q_i+1}^{d_j} \xi_{ij} \cdot \prod_{\substack{i=1\\ S_i\in\Theta}}^{n} \sum_{(\alpha,r)\in\mathcal{A}ct(S_i)} r$$

where $1/G$ is a normalising constant.

Proof

The proof proceeds by considering sets of global balance equations, and showing that the proposed solution does in fact satisfy them. Each state of an underlying process is identified with a process algebra term; this is legitimate since each model is a continuous time Markov chain, and a one-to-one correspondence exists between model states and process algebra terms.

First, consider the global balance equations for a sequential process S_i in isolation. There is no queueing discipline to restrict its behaviour, and so

- the flow out of the current state is due to individual activities enabled by S_i; and
- the flow into the current state is due to all activities enabled by predecessors of S_i which may lead to S_i.

The global balance equations are given by

$$\begin{aligned} \text{flux out of } S_i &= \pi_i(S_i) \cdot \sum_{(\alpha,r)\in\mathcal{A}ct(S_i)} r \\ \text{flux into } S_i &= \sum_{\{^\diamond S_i\}} \pi_i(^\diamond S_i) \cdot \sum_{\{(\alpha,r)\,:\,^\diamond S_i \longrightarrow S_i\}} r \end{aligned} \tag{4.3.12}$$

Now consider the general case of global balance for P. For $0 \leq i \leq N$, the ith queue will contain $0 \leq q_i \leq d_i$ processes (subject to the restriction that $\sum_{i=0}^{N} q_i \leq n$). The flow out of P is given by

- activities enabled by each $S_i \notin \Theta$, that is individual activities enabled by components which are not currently queueing; and
- for each queue j, the flow out due to the process currently at the head of the queue (if $q_j \neq 0$).

The flow into P is due to flow from predecessors ${}^{\diamond}P$; it consists of

- individual activities enabled by a predecessor of S_i, ${}^{\diamond}S_i \notin \Theta$, where ${}^{\diamond}P$ differs from P at the ith sequential component only; this flow does not involve any of the queueing disciplines;
- for each queue j, the flow from ${}^{\diamond}P$ which represents a system identical to P except that the jth queue contains one more process, and where ${}^{\diamond}P \longrightarrow P$ is due to the extra process leaving the queue; note that in this case, the extra process S_i must not be queueing in the current state P—recall a process cannot enter one queue by leaving another;
- for each queue j, the flow from ${}^{\diamond}P$ which represents a system identical to P except that the jth queue contains one less process, and where ${}^{\diamond}P \longrightarrow P$ is due to a sequential process S_i joining the jth queue; note that in this case, the extra process S_i must not be queueing in the predecessor state ${}^{\diamond}P$, by the same rule as in the point above.

Notice that by construction of $P \equiv Q_{\chi}(S_1, \ldots, S_n)$, there is no direct cooperation between any of the S_i, and thus such terms do not contribute to the global balance equations. Using the notation introduced earlier, the equations can be stated formally as follows:

$$\text{flux out of } P = \pi(Q_{\chi}(S_1, \ldots, S_n)).\left(\sum_{\substack{i=1 \\ S_i \notin \Theta}}^{n} \sum_{(\alpha,r) \in \mathcal{A}ct(S_i)} r + \sum_{i=1}^{N} \xi_{iq_i} \right) \qquad (4.3.13)$$

$$\text{flux into } P = \sum_{\substack{i=1\\ S_i\notin\Theta}}^{n} \sum_{\{{}^\diamond S_i\,:\,{}^\diamond S_i\notin\Theta\}} \pi(Q_\chi(S_1,\ldots,{}^\diamond S_i,\ldots,S_n)). \sum_{\{(\alpha,r)\,:\,{}^\diamond S_i \xrightarrow{(\alpha,r)} S_i\}} r$$

$$+ \sum_{\substack{i=1\\ S_i\notin\Theta}}^{n} \sum_{\{{}^\diamond S_i\,:\,\exists j\,:\,i=H_j\}} \pi(Q_\chi(S_1,\ldots,{}^\diamond S_i,\ldots,S_n))\cdot\xi_{jq_j+1}$$

$$+ \sum_{\substack{j=1\\ T_j \text{is defined}}}^{N} \sum_{\{{}^\diamond S_{T_j}\}} \pi(Q_\chi(S_1,\ldots,{}^\diamond S_{T_j},\ldots,S_n)). \sum_{\{(\alpha,r)\,:\,{}^\diamond S_{T_j} \xrightarrow{(\alpha,r)} S_{T_j}\}} r$$

In order to prove the theorem, the proposed solution is substituted for the steady-state distribution $\pi(\cdot)$, and it is shown that for an arbitrary state P of $Q_\chi(S_1,\ldots,S_n)$,

$$\text{flux out of } P = \text{flux into } P$$

First, consider expanding the flux out of P.

$$\text{flux out of } P = \left(\frac{1}{G}\prod_{i=1}^{n}\pi_i(S_i)\cdot\prod_{i=1}^{N}\prod_{j=q_i+1}^{d_i}\xi_{ij}\cdot\prod_{\substack{i=1\\ S_i\in\Theta}}^{n}\sum_{(\alpha,r)\in\mathcal{A}ct(S_i)} r\right)\cdot\left(\sum_{\substack{i=1\\ S_i\notin\Theta}}^{n}\sum_{(\alpha,r)\in\mathcal{A}ct(S_i)} r+\sum_{i=1}^{N}\xi_{iq_i}\right) \tag{4.3.14}$$

This must be matched by the flux into P.

$$\text{flux into } P = \sum_{\substack{i=1\\ S_i\notin\Theta}}^{n} \sum_{\{{}^\diamond S_i\,:\,{}^\diamond S_i\notin\Theta\}} \boldsymbol{A}\cdot \sum_{\{(\alpha,r)\,:\,{}^\diamond S_i \xrightarrow{(\alpha,r)} S_i\}} r + \sum_{\substack{i=1\\ S_i\notin\Theta}}^{n} \sum_{\{{}^\diamond S_i\,:\,\exists j\,:\,i=H_j\}} \boldsymbol{B}\cdot\xi_{jq_j+1}$$

$$+ \sum_{\substack{j=1\\ T_j \text{is defined}}}^{N} \sum_{\{{}^\diamond S_{T_j}\}} \boldsymbol{C}\cdot \sum_{\{(\alpha,r)\,:\,{}^\diamond S_{T_j} \xrightarrow{(\alpha,r)} S_{T_j}\}} r \tag{4.3.15}$$

where

$$\begin{aligned}
\boldsymbol{A} &= \frac{1}{G}\prod_{\substack{j=1\\ j\neq i}}^{n}\pi_j(S_j)\cdot\pi_i({}^\diamond S_i)\cdot\prod_{j=1}^{N}\prod_{k=q_j+1}^{d_j}\xi_{jk}\cdot\prod_{\substack{j=1\\ S_j\in\Theta}}^{n}\sum_{(\alpha,r)\in\mathcal{A}ct(S_j)} r\\
\boldsymbol{B} &= \frac{1}{G}\prod_{\substack{k=1\\ k\neq i}}^{n}\pi_k(S_k)\cdot\pi_i({}^\diamond S_i)\cdot\prod_{\substack{k=1\\ k\neq j}}^{N}\prod_{l=q_k+1}^{d_k}\xi_{kl}\cdot\prod_{l=q_j+2}^{d_j}\xi_{jl}\cdot\prod_{\substack{k=1\\ S_k\in\Theta}}^{n}\sum_{(\alpha,r)\in\mathcal{A}ct(S_k)} r\cdot\sum_{(\alpha,r)\in\mathcal{A}ct({}^\diamond S_i)} r\\
\boldsymbol{C} &= \frac{1}{G}\prod_{\substack{i=1\\ i\neq T_j}}^{n}\pi_i(S_i)\cdot\pi_{T_j}({}^\diamond S_{T_j})\cdot\prod_{\substack{k=1\\ k\neq j}}^{N}\prod_{l=q_k+1}^{d_k}\xi_{kl}\cdot\prod_{l=q_j}^{d_j}\xi_{jl}\cdot\prod_{\substack{k=1\\ k\neq T_j\\ S_k\in\Theta}}^{n}\sum_{(\alpha,r)\in\mathcal{A}ct(S_k)} r
\end{aligned} \tag{4.3.16}$$

First, observe that when a process ${}^\diamond S_i$ may perform its queue activity (evolving into S_i), it does not possess the potential to perform any other activity, and therefore

$$\mathcal{A}ct({}^\diamond S_i) = \{(\alpha,r)\,:\,{}^\diamond S_i \xrightarrow{(\alpha,r)} S_i\} \implies \sum_{(\alpha,r)\in\mathcal{A}ct({}^\diamond S_i)} r = \sum_{\{(\alpha,r)\,:\,{}^\diamond S_i \xrightarrow{(\alpha,r)} S_i\}} r \tag{4.3.17}$$

Now consider the first term of Equation (4.3.15). For each S_i not currently queueing, by the individual balance shown in Equation (4.3.12), it is possible to simplify this term to the following.

$$\begin{aligned}
&\sum_{\substack{i=1\\ S_i\notin\Theta}}^{n} \sum_{\{{}^\diamond S_i\,:\,{}^\diamond S_i\notin\Theta\}} \left(\frac{1}{G}\prod_{\substack{j=1\\ j\neq i}}^{n}\pi_j(S_j)\cdot\prod_{j=1}^{N}\prod_{k=q_j+1}^{d_j}\xi_{jk}\cdot\prod_{\substack{j=1\\ S_j\in\Theta}}^{n}\sum_{(\alpha,r)\in\mathcal{A}ct(S_j)} r\right)\cdot\pi_i(S_i)\cdot\sum_{(\alpha,r)\in\mathcal{A}ct(S_i)} r\\
&=\sum_{\substack{i=1\\ S_i\notin\Theta\\ {}^\diamond S_i\notin\Theta}}^{n}\left(\frac{1}{G}\prod_{j=1}^{n}\pi_j(S_j)\cdot\prod_{j=1}^{N}\prod_{k=q_j+1}^{d_j}\xi_{jk}\cdot\prod_{\substack{j=1\\ S_j\in\Theta}}^{n}\sum_{(\alpha,r)\in\mathcal{A}ct(S_j)} r\right)\cdot\sum_{(\alpha,r)\in\mathcal{A}ct(S_i)} r
\end{aligned} \tag{4.3.18}$$

Secondly, the second term of Equation (4.3.15) may be simplified. By construction of the steady-state solution, it can be readily seen that the flow rate, ξ_{jq_j}, can be absorbed into a product term; and once again, by the individual balance given in Equation (4.3.12), the whole term can be rewritten as follows.

$$\begin{aligned}
\sum_{\substack{i=1\\ S_i\notin\Theta}}^{n}\sum_{\{{}^\diamond S_i\,:\,\exists j\,:\,i=H_j\}}\boldsymbol{A}\cdot\xi_{jq_j+1} &= \sum_{\substack{i=1\\ S_i\notin\Theta}}^{n}\sum_{\{{}^\diamond S_i\,:\,\exists j\,:\,i=H_j\}}\boldsymbol{B}\cdot\sum_{(\alpha,r)\in\mathcal{A}ct(S_i)} r\\
&= \sum_{\substack{i=1\\ S_i\notin\Theta\\ \{{}^\diamond S_i\,:\,\exists j\,:\,i=H_j\}}}^{n}\boldsymbol{C}\cdot\sum_{(\alpha,r)\in\mathcal{A}ct(S_i)} r
\end{aligned} \tag{4.3.19}$$

where

$$\begin{aligned}
\boldsymbol{A} &= \frac{1}{G}\prod_{\substack{k=1\\ k\neq i}}^{n}\pi_k(S_k)\cdot\pi_i({}^\diamond S_i)\cdot\prod_{\substack{k=1\\ k\neq j}}^{N}\prod_{l=q_k+1}^{d_k}\xi_{kl}\cdot\prod_{l=q_j+2}^{d_j}\xi_{jl}\cdot\prod_{\substack{k=1\\ S_k\in\Theta}}^{n}\sum_{(\alpha,r)\in\mathcal{A}ct(S_k)} r\cdot\sum_{(\alpha,r)\in\mathcal{A}ct({}^\diamond S_i)} r\\
\boldsymbol{B} &= \frac{1}{G}\prod_{\substack{k=1\\ k\neq i}}^{n}\pi_k(S_k)\cdot\pi_i({}^\diamond S_i)\cdot\prod_{\substack{k=1\\ k\neq j}}^{N}\prod_{l=q_k+1}^{d_k}\xi_{kl}\cdot\prod_{l=q_j+1}^{d_j}\xi_{jl}\cdot\prod_{\substack{k=1\\ S_k\in\Theta}}^{n}\sum_{(\alpha,r)\in\mathcal{A}ct(S_k)} r\\
\boldsymbol{C} &= \frac{1}{G}\prod_{k=1}^{n}\pi_k(S_k)\cdot\prod_{k=1}^{N}\prod_{l=q_k+1}^{d_k}\xi_{kl}\cdot\prod_{\substack{k=1\\ S_k\in\Theta}}^{n}\sum_{(\alpha,r)\in\mathcal{A}ct(S_k)} r
\end{aligned} \tag{4.3.20}$$

However, note that for i such that $1 \leq i \leq n$, if the ith component S_i is currently not queueing, then the flow into the current state is given by the flow out of a previous state due to the ith component being present at the head of a queue, and leaving the queue; or by the ith component making a transition to S_i, involving no queue. Therefore, Equations (4.3.18) and (4.3.19) can be amalgamated to give the following.

$$\sum_{\substack{i=1\\ S_i\notin\Theta}}^{n}\left(\frac{1}{G}\prod_{k=1}^{n}\pi_k(S_k)\cdot\prod_{k=1}^{N}\prod_{l=q_k+1}^{d_k}\xi_{kl}\cdot\prod_{\substack{k=1\\ S_k\in\Theta}}^{n}\sum_{(\alpha,r)\in\mathcal{A}ct(S_k)} r\right)\cdot\sum_{(\alpha,r)\in\mathcal{A}ct(S_i)} r \tag{4.3.21}$$

Next, consider the third term of Equation (4.3.15). First of all, the local balance of Equation (4.3.12) can again be exploited, this time for each sequential component S_i queueing in the current state, but not queueing in the previous state. Then the sum of activities giving the flow out can be amalgamated into the main product term; it is replaced by removing ξ_{jq_j} and factoring it out to the right hand side of the expression.

$$
\begin{aligned}
&\sum_{\substack{j=1\\ T_j\text{is defined}}}^{N} \sum_{\{^\diamond S_{T_j}\}} \boldsymbol{D} \cdot \sum_{\{(\alpha,r)\,:\,^\diamond S_{T_j} \xrightarrow{(\alpha,r)} S_{T_j}\}} r \\
&= \sum_{\substack{j=1\\ T_j\text{is defined}}}^{N} \left(\frac{1}{G} \prod_{\substack{i=1\\ i\neq T_j}}^{n} \pi_i(S_i) \cdot \prod_{\substack{k=1\\ k\neq j}}^{N} \prod_{l=q_k+1}^{d_k} \xi_{kl} \cdot \prod_{l=q_j}^{d_j} \xi_{jl} \cdot \prod_{\substack{k=1\\ k\neq T_j\\ S_k\in\Theta}}^{n} \sum_{(\alpha,r)\in\mathcal{A}ct(S_k)} r \right) \cdot \pi_{T_j}(S_{T_j}) \cdot \sum_{(\alpha,r)\in\mathcal{A}ct(S_{T_j})} r \\
&= \sum_{\substack{j=1\\ T_j\text{is defined}}}^{N} \left(\frac{1}{G} \prod_{i=1}^{n} \pi_i(S_i) \cdot \prod_{\substack{k=1\\ k\neq j}}^{N} \prod_{l=q_k+1}^{d_k} \xi_{kl} \cdot \prod_{l=q_j+1}^{d_j} \xi_{jl} \cdot \prod_{\substack{k=1\\ S_k\in\Theta}}^{n} \sum_{(\alpha,r)\in\mathcal{A}ct(S_k)} r \right) \cdot \xi_{jq_j} \\
&= \sum_{j=1}^{N} \left(\frac{1}{G} \prod_{i=1}^{n} \pi_i(S_i) \cdot \prod_{k=1}^{N} \prod_{l=q_k+1}^{d_k} \xi_{kl} \cdot \prod_{\substack{k=1\\ S_k\in\Theta}}^{n} \sum_{(\alpha,r)\in\mathcal{A}ct(S_k)} r \right) \cdot \xi_{jq_j}
\end{aligned}
\tag{4.3.22}
$$

where

$$
\boldsymbol{D} = \frac{1}{G} \prod_{\substack{i=1\\ i\neq T_j}}^{n} \pi_i(S_i) \cdot \pi_{T_j}(^\diamond S_{T_j}) \cdot \prod_{\substack{k=1\\ k\neq j}}^{N} \prod_{l=q_k+1}^{d_k} \xi_{kl} \cdot \prod_{l=q_j}^{d_j} \xi_{jl} \cdot \prod_{\substack{k=1\\ k\neq T_j\\ S_k\in\Theta}}^{n} \sum_{(\alpha,r)\in\mathcal{A}ct(S_k)} r \tag{4.3.23}
$$

The final step in the simplification is due to the fact that $\xi_{iq_i} = 0$ if $q_i = 0$ by definition. From here, it is clear that in both Equations (4.3.21) and (4.3.22), each central bracketed term does not depend on the index in the outer summation. These terms are added together to give the following.

$$
\begin{aligned}
&\sum_{\substack{i=1\\ S_i\notin\Theta}}^{n} \left(\frac{1}{G} \prod_{k=1}^{n} \pi_k(S_k) \cdot \prod_{k=1}^{N} \prod_{l=q_k+1}^{d_k} \xi_{kl} \cdot \prod_{\substack{k=1\\ S_k\in\Theta}}^{n} \sum_{(\alpha,r)\in\mathcal{A}ct(S_k)} r \right) \cdot \sum_{(\alpha,r)\in\mathcal{A}ct(S_i)} r \\
&+ \sum_{j=1}^{N} \left(\frac{1}{G} \prod_{i=1}^{n} \pi_i(S_i) \cdot \prod_{k=1}^{N} \prod_{l=q_k+1}^{d_k} \xi_{kl} \cdot \prod_{\substack{k=1\\ S_k\in\Theta}}^{n} \sum_{(\alpha,r)\in\mathcal{A}ct(S_k)} r \right) \cdot \xi_{jq_j} \\
&= \left(\frac{1}{G} \prod_{i=1}^{n} \pi_i(S_i) \cdot \prod_{i=1}^{N} \prod_{j=q_i+1}^{d_i} \xi_{ij} \cdot \prod_{\substack{i=1\\ S_i\in\Theta}}^{n} \sum_{(\alpha,r)\in\mathcal{A}ct(S_i)} r \right) \cdot \left(\sum_{\substack{i=1\\ S_i\notin\Theta}}^{n} \sum_{(\alpha,r)\in\mathcal{A}ct(S_i)} r + \sum_{i=1}^{N} \xi_{iq_i} \right)
\end{aligned}
\tag{4.3.24}
$$

This final term equals the flux out of P presented in Equation (4.3.14), and so the theorem is proven. □

Now that the general form of the steady-state solution has been exhibited for models built with the queueing combinator, it is straightforward to examine further sets of balance equations. The main interest in this chapter is insensitivity, and the next result looks at *insensitivity balance* equations for particular activities present in queueing combinator models.

Theorem 4.3.2. *Given a PEPA model $P \equiv Q_{\chi}(S_1, \ldots, S_n)$, P is insensitive to each activity in $\overrightarrow{\mathcal{A}ct}(S_i)$, for $1 \leq i \leq n$, with the exception of those activities in the set $\bigcup_{i=1}^{N} A_i \cup e_{A_i}(S_i)$.*

Proof

This proof ignores consideration of all activities involved in queueing, since the aim is to show which activities of the original sequential processes may be assumed to be generally distributed, without modification—the queueing combinator insists that the queue activities of the original process's are altered such that their duration depends on the current lengths of the queues in which the processes reside.

The proof proceeds by constructing the insensitivity balance equations for the required activities, and using the general form of solution for these models to show the equations are satisfied. Since it is the case that in the GSMP translation of a queueing model, active elements are multisets of activities, the proof actually considers the insensitivity of a collection of mutually disabling activities; the ramifications of this fact are explained after the presentation of the proof.

As given in Corollary 3, in order for a PEPA process to exhibit insensitivity balance with respect to an active element s, it is necessary that for every derivative P in which s is enabled, the flux out of P due to the completion of s is equal to the flux into P from derivatives in which s is not enabled, that is the flux into P enabling s. Fix a state $P \equiv Q_{\chi}(S_1, \ldots, S_n)$. An active element e will correspond to a multiset of activities enabled by one of the constituent sequential components; let this component be S_i. Therefore it is implicitly assumed that at least one of the constituent sequential components is not currently queueing. The insensitivity balance equations are formally stated as follows.

$$\text{flux out of } P \text{ due to death of } s = \pi(Q_{\chi}(S_1, \ldots, S_n)) \cdot \sum_{(\alpha,r) \in \mathcal{A}ct(S_i)} r \qquad (4.3.25)$$

$$\text{flux into } P \text{ due to birth of } s = \sum_{\{{}^\diamond S_i \,:\, \exists j \,:\, i=H_j\}} \pi(Q_\chi(S_1,\ldots,{}^\diamond S_i,\ldots,S_n)) \cdot \xi_{jq_j+1}$$
$$+ \sum_{\{{}^\diamond S_i \,:\, {}^\diamond S_i \notin \Theta\}} \pi(Q_\chi(S_1,\ldots,{}^\diamond S_i,\ldots,S_n)) \cdot \sum_{\{(\alpha,r)\,:\, {}^\diamond S_i \xrightarrow{(\alpha,r)} S_i\}} r \tag{4.3.26}$$

Notice that there can be no flux into P leading to the birth of s due to any sequential components entering a queue; it is assumed that in the current state S_i is not queueing, and the evolution of every other S_j, whether restrained by the queueing discipline or not, cannot lead to the birth of s. Substituting the solution exhibited in Theorem 4.3.1 results in the following.

$$\text{flux out of } P \text{ due to death of } s = \boldsymbol{A} \cdot \sum_{(\alpha,r)\in \mathcal{A}ct(S_i)} r \tag{4.3.27}$$

$$\text{flux into } P \text{ due to birth of } s = \sum_{\{{}^\diamond S_i \,:\, \exists j \,:\, i=H_j\}} \boldsymbol{B} \cdot \xi_{jq_j+1}$$
$$+ \sum_{\{{}^\diamond S_i \,:\, {}^\diamond S_i \notin \Theta\}} \boldsymbol{C} \cdot \sum_{\{(\alpha,r)\,:\, {}^\diamond S_i \xrightarrow{(\alpha,r)} S_i\}} r \tag{4.3.28}$$

where

$$\boldsymbol{A} = \frac{1}{G}\prod_{i=1}^{n} \pi_i(S_i) \cdot \prod_{i=1}^{N}\prod_{j=q_i+1}^{d_i} \xi_{ij} \cdot \prod_{\substack{i=1\\ S_i\in\Theta}}^{n} \sum_{(\alpha,r)\in\mathcal{A}ct(S_i)} r$$

$$\boldsymbol{B} = \frac{1}{G}\prod_{\substack{k=1\\ k\neq i}}^{n} \pi_k(S_k) \cdot \pi_i({}^\diamond S_i) \cdot \prod_{\substack{k=1\\ k\neq j}}^{N}\prod_{l=q_k+1}^{d_k} \xi_{kl} \cdot \prod_{l=q_j+2}^{d_j} \xi_{jl} \cdot \prod_{\substack{k=1\\ S_k\in\Theta}}^{n} \sum_{(\alpha,r)\in\mathcal{A}ct(S_k)} r \cdot \sum_{(\alpha,r)\in\mathcal{A}ct({}^\diamond S_i)} r$$

$$\boldsymbol{C} = \frac{1}{G}\prod_{\substack{j=1\\ j\neq i}}^{n} \pi_j(S_j) \cdot \pi_i({}^\diamond S_i) \cdot \prod_{j=1}^{N}\prod_{k=q_j+1}^{d_j} \xi_{jk} \cdot \prod_{\substack{j=1\\ S_j\in\Theta}}^{n} \sum_{(\alpha,r)\in\mathcal{A}ct(S_j)} r \tag{4.3.29}$$

However, immediately the two terms in Equation (4.3.28) can be amalgamated, beginning by absorbing the outer queue dependent rate into the solution term. This simplification leads to the following.

flux into P due to birth of s

$$= \sum_{\{{}^\diamond S_i \,:\, \exists j \,:\, i=H_j\}} \left(\frac{1}{G}\prod_{\substack{k=1\\ k\neq i}}^{n} \pi_k(S_k) \cdot \pi_i({}^\diamond S_i) \cdot \prod_{k=1}^{N}\prod_{l=q_k+1}^{d_k} \xi_{kl} \cdot \prod_{\substack{k=1\\ S_k\in\Theta}}^{n} \sum_{(\alpha,r)\in\mathcal{A}ct(S_k)} r\right) \cdot \sum_{(\alpha,r)\in\mathcal{A}ct({}^\diamond S_i)} r$$
$$+ \sum_{\{{}^\diamond S_i \,:\, {}^\diamond S_i \notin \Theta\}} \left(\frac{1}{G}\prod_{\substack{j=1\\ j\neq i}}^{n} \pi_j(S_j) \cdot \pi_i({}^\diamond S_i) \cdot \prod_{j=1}^{N}\prod_{k=q_j+1}^{d_j} \xi_{jk} \cdot \prod_{\substack{j=1\\ S_j\in\Theta}}^{n} \sum_{(\alpha,r)\in\mathcal{A}ct(S_j)} r\right) \cdot \sum_{\{(\alpha,r)\,:\, {}^\diamond S_i \xrightarrow{(\alpha,r)} S_i\}} r \tag{4.3.30}$$

The first term of Equation (4.3.30) accounts for the departure from predecessor states by ${}^{\diamond}S_i$ leaving the head of a queue. By assumption, the queue activity is the only activity enabled by ${}^{\diamond}S_i$ in this position, and so by Equation (4.3.17), the rate of departure can instead be given by $\sum_{\{(\alpha,r)\,:\,{}^{\diamond}S_i \xrightarrow{(\alpha,r)} S_i\}} r$. This makes the correspondence between the two terms clear, and they can be amalgamated to give Equation (4.3.31).

$$\text{flux into } P \text{ due to birth of } s$$
$$= \sum_{\{{}^{\diamond}S_i\}} \left(\frac{1}{G} \prod_{\substack{k=1\\k\neq i}}^{n} \pi_k(S_k) \cdot \pi_i({}^{\diamond}S_i) \cdot \prod_{\substack{k=1\\k\neq j}}^{N} \prod_{l=q_k+1}^{d_k} \xi_{kl} \cdot \prod_{\substack{k=1\\S_k\in\Theta}}^{n} \sum_{(\alpha,r)\in\mathcal{A}ct(S_k)} r \right) \cdot \sum_{\{(\alpha,r)\,:\,{}^{\diamond}S_i \xrightarrow{(\alpha,r)} S_i\}} r \tag{4.3.31}$$

Finally, the local balance given in Equation (4.3.12) can be exploited for component S_i, leading to the following term.

$$\text{flux into } P \text{ due to birth of } s$$
$$= \left(\frac{1}{G} \prod_{k=1}^{n} \pi_k(S_k) \cdot \prod_{\substack{k=1\\k\neq j}}^{N} \prod_{l=q_k+1}^{d_k} \xi_{kl} \cdot \prod_{\substack{k=1\\S_k\in\Theta}}^{n} \sum_{(\alpha,r)\in\mathcal{A}ct(S_k)} r \right) \cdot \sum_{(\alpha,r)\in\mathcal{A}ct(S_i)} r \tag{4.3.32}$$

This is equal to the flux out of P due to the death of s as given in Equation (4.3.25), and so insensitivity balance holds for all such active elements s. □

4.3.4 An Example Model

A simple example is now given to illustrate the structure of models generated using the new combinator. Consider the following definition of a simple processor.

$$\begin{aligned} CPU_i &\stackrel{\text{def}}{=} (\texttt{compute}, r\cdot\delta).(\texttt{claim}_\texttt{i}, \lambda).(\texttt{release}_\texttt{i}, \mu).CPU_i \\ &+ (\texttt{compute}, r\cdot(1-\delta)).(\texttt{sleep}, s).CPU_i \end{aligned} \tag{4.3.33}$$

Its purpose is to repeatedly perform work, and then either idle with high probability, or attempt to access a shared memory resource with low probability. Now consider the following model:

$$\begin{aligned} &MultiProc \stackrel{\text{def}}{=} Q_{A,\xi}(CPU_1, \ldots, CPU_n) \\ &\text{where } A = \{\texttt{release}_\texttt{i} : 1 \leq i \leq n\} \end{aligned} \tag{4.3.34}$$

This defines a model where n processors compete for access to one shared memory resource. If a processor is using the memory at the time another attempts to access it, the latter processor is blocked, and forced to queue. If i processors are currently queued, the rate at which the CPU at the head of the queue performs its `release` activity is given by ξ_i. In this example, it may make sense to set $\xi_i = \xi_j$ for all i, j. Theorem 4.3.2

then guarantees that all activities except $\{\mathtt{claim_i}, \mathtt{release_i} : 1 \leq i \leq n\}$ are insensitive to the form of their distributions. For example, the model will be guaranteed to have the same steady-state solution if any `sleep` activity is instead considered to be uniformly distributed between two positive values, with mean $1/s$.

4.4 Discussion

An interesting fact about this insensitivity result is that the given syntactic characterisation is distinct from the known criteria for SPA models which generate product form solutions (over submodels). A model which exhibits product form over submodels may be restricted in form, but allows for an efficient solution. The components of a model can be solved in isolation, and then combined to give a solution for the model as a whole. This approach avoids the computational burden involved in typical non-compositional techniques.

Harrison and Hillston [33] explore SPA structures which give rise to *quasi-reversible* (QR) models. Consider a stochastic process whose state is represented by a countable set of elements. An input process causes the stochastic process to change state by changing the state of one of the elements; or an output process may reverse this state change by converting the element back. The process is QR if and only if for all times t, the current state of the process is independent of the input process after t, and the output process before t. The authors show that a PEPA model consisting of a cooperation of processes has a QR stochastic process if considered pairwise, each of the cooperations form a *channel*. This means that a cooperation between two components leads to a derivative for which a cooperation is available to return the model to the original state.

Later work, by Hillston and Thomas [46], takes advantage of a generalisation made by Boucherie [10], where several distinct product form results for SPN are shown to be special cases of a simple exclusion mechanism for a product process. In their setting, subcomponents compete over resources, with the restriction that if a subcomponent possesses a resource which another would like now, or at some point in its lifespan, then the rival is blocked until the resource is freed. The syntactic form of the PEPA models which generate such product form solutions look similar to the form presented in this chapter. Further work is required to determine if the two approaches will easily combine. However, resources claimed in the Boucherie setting are denied to others, which if they wish to use the resource now, or at some future point in their behaviours, must block. Clearly, the arbiter processes present in a queueing discipline model do not act as resources in this way, since they do not prevent the evolution of other components not currently queueing. Moreover, note that our approach differs in that the structure is described entirely in terms of a new derived combinator, which allows the construction of models with the stated insensitivity property.

Sereno [68] takes a different approach to finding product form solutions for PEPA models. He finds conditions on the *traffic equations* that guarantee the existence of a product form solution. The traffic equations for a PEPA model are analogous to those used in queueing theory, and are formed from the *routing process* generated by considering activities as states themselves. Again, the approach presented in this chapter is in a different direction, in that an operator is defined which *guarantees* a product form, and with which insensitive models may be constructed. Note that the existing work on product form for process algebra contains many common ideas, and it is conjectured that it will be possible to amalgamate thus-far separate ideas under an encompassing theory.

The product form solution for queueing discipline models strongly suggests a link to the well-known class of BCMP network queueing models [4]. This similarity is worth exploring further and formalising. The BCMP theorem states that queueing networks with nodes which can be classified as

- First come first served (FCFS); or
- Processor sharing (PS); or
- Infinite server (IS); or
- Last come first served–Premptive resume (LCFS-PR)

and which also contain multi-class traffic have a product form solution for the steady-state probability distribution of the node states. A comparison with the structure of the models presented in this chapter suggests that it may be possible to view them too as networks of queues; each bottleneck point may represent a FCFS node, where the service time for each customer is identically distributed and dependent on the queue length; and the non-queueing activities could be seen to represent IS nodes, where notionally, each customer is delayed only by a service time. This comparison has not been made in any detail, and remains as interesting future work. However it may suggest directions in which to extend our construction—an obvious first step is to consider the PEPA analogues of PS and LCFS-PR nodes, and investigate whether they may be simply integrated into the construction as it stands. A tentative first attempt at a processor sharing node is given in Equation (4.4.1).

$$\begin{aligned} PS_B &\stackrel{\text{def}}{=} \sum_{P \in B} (\alpha_P, r_P/n).PS_{B\setminus\{P\}} \\ &+ \sum_{P \notin B} (\beta_P, \top).PS_{B\cup\{P\}} \\ &\quad \text{where } n = |B| \end{aligned} \tag{4.4.1}$$

Processes are captured by cooperation with an activity of type β; then are forced to leave the node by cooperation with an activity that guarantees to reduce the rate at which they leave by a constant dependent on the current number of processes present at the PS node. This is built in an attempt to be analogous with the BCMP definition. The next step would be to generalise the definition of the queueing combinator and the current product form solution term. Although this work remains to be done in its entirety, it is the author's hope that this might provide a beginning for a comparison of process algebra and queueing networks in general.

It is worth noting that the queueing structure presented in this chapter by no means captures all insensitive process algebra structures. This can be seen via a simple example. Consider the following definitions and process.

$$\begin{aligned} P &\stackrel{\text{def}}{=} (\alpha, r).P' \\ Q &\stackrel{\text{def}}{=} (\alpha, s).Q' \\ R &\stackrel{\text{def}}{=} P \underset{\{\alpha\}}{\bowtie} P \underset{\{\alpha\}}{\bowtie} Q \underset{\{\alpha\}}{\bowtie} Q \end{aligned} \tag{4.4.2}$$

R consists of a four-way cooperation between processes. This ensures that if one of the subcomponents wishes to perform an α activity, it must wait for all others. A result of this structure is that the transition system for R contains only one transition corresponding to this α activity; in effect, the state space becomes a bottleneck at this point, where one transition represents the cooperative progress of each subcomponent. Therefore, viewing the model as a CTMC, there is one state in which this transition is enabled, and in the successor state, the activity is completed and thus disabled. It is simple to see that global balance must equal insensitivity balance at this point. The queueing combinator does not capture models with this form of cooperation.

To summarise, this chapter presents a new derived combinator for PEPA. This combinator is used to construct models of sequential processes which synchronise according to a queueing discipline. It is shown that despite such synchronisation, the insensitivity of residence time in particular states of the model is guaranteed, and thus insensitivity of PEPA activities is also guaranteed. The modeller may notionally employ a non-exponential distribution to model the duration of a particular set of activities.

Chapter 5

Insensitivity Conditions for PEPA Models

5.1 Introduction

This chapter studies more general conditions for insensitivity within PEPA models. The work is independent of the GSMP model, and can be viewed as an extension of earlier work by Hillston on *weak isomorphism* [44]. A formal extension to the language of PEPA is presented, which allows for generally distributed activities, and it is shown that the models are insensitive to these activities under certain balance conditions. In this way, it is shown that any structural condition which guarantees the form of balance exhibited here will guarantee insensitivity of a particular set of activities, or at least insensitivity of residence time in particular states of the process. An original motivation for this work was to give a more general performance-preserving equivalence relation for PEPA.

The chapter is organised as follows. Section 5.2 introduces the language gPEPA, a formal extension to PEPA which allows generally distributed activities in a limited way. Next, Sections 5.2.1–5.2.3 deal with the construction of faithful models for a gPEPA process; the performance model is also a continuous time Markov chain. Section 5.3 then describes the sets of balance equations relevant to both global balance, and insensitivity balance. This leads to the main proposition of the chapter, in which it is shown that particular activities may be generally distributed, if a set of insensitivity balance equations are satisfied. Section 5.4 discusses the ramifications of this result.

5.2 Extending PEPA with General Distributions

The work presented here extends PEPA to incorporate activities which are generally distributed. This is done in a restricted way, under conditions which are described in this section. The work extends that carried out by Hillston, described in Chapter 6 of [44]. The extension to PEPA is called gPEPA and an example of a gPEPA process is

$$\overline{P} \stackrel{\mathrm{def}}{=} (\alpha, r).(\beta, X).\overline{P}$$

PEPA specifies that all activities are exponentially distributed. The time-to-complete of such an activity is characterised by one real number, the *rate*, which is the parameter of the distribution. gPEPA introduces generally distributed activities. The parameter X denotes a random variable governing the length of time which will pass before (β, X) can or will occur (and not the rate at which (β, X) can or will occur). The distribution function associated with X is denoted by $F_X(\cdot)$ such that $F_X(t) = \Pr(X \leq t)$. The behaviour of $\overline{P}$ above is as one might expect; α is performed at rate r followed by β after X units of time; then this behaviour repeats. To make sense in this continuous time context, random variables which may take non-positive values are disallowed. The set of all such random variables is denoted by *PRV*. To aid in distinguishing the two, a gPEPA process name will always appear with an overbar, e.g. $\overline{P}$. The restrictions and assumptions on, and the consequences of, the use of general distributions in gPEPA are now described.

- *A generally distributed activity may not synchronise with any other activity.* This is a strong condition, and means that any synchronisation (cooperation) between processes must take place on exponential activities. Note, however, that these processes may still *enable* generally distributed activities. Consequently, if a sequential component $\overline{S_i}$ of a process completes a lifetime of a generally distributed activity, then any difference in the derivative vector of components must only be in the ith place.

- *A synchronisation of sequential components may not lead to a state where more than one sequential component newly enables a generally distributed activity.* This restriction does not prevent two generally distributed activities being newly enabled by the same sequential component, but then analysing transition probabilities is a separate, and of course difficult, issue.

In order to understand the behaviour of gPEPA processes, a semantics is necessary. A feature of this semantics must be that the remaining lifetimes of generally distributed activities are retained upon transitions—the memoryless property of the exponential is not now available in general. The approach taken in this chapter is to use a structured operational semantics which induces exponential and probabilistic transition relations. The rôle of probabilistic transitions will be explained in the next section. Furthermore, the relationship between a gPEPA model and a (potentially infinite-state) continuous time Markov chain will then be detailed.

5.2.1 Representing General Distributions

In this chapter, general distributions are represented by the *Erlang mixture*, a probabilistic sum over a set of convolutions of exponential distributions. The *k-stage Erlang* distribution, consisting of the convolution of k consecutive exponential distributions each with parameter ν, is denoted $E(k,\nu)$. An Erlang mixture distribution is a (generally infinite) probabilistic sum of Erlang distributions, and is represented by

$$\sum_{k=1}^{\infty} p(k)E(k,\nu)$$

The function $p(\cdot)$ forms a probability distribution over the event space $\{i \in \mathbb{Z}, i \geq 1\}$, and therefore the probability an Erlang mixture will begin with k stages is given by $p(k)$. It has been shown that the Erlang mixture distribution can arbitrarily closely approximate any general probability distribution [24] (so long as the distribution has a non-negative support set). Furthermore, it may then be possible to obtain results for arbitrary general distributions. The following lemma establishes a useful result that will be employed later.

Lemma 5.2.1. *Consider a renewal process with mean renewal time distributed as* $\sum_{k=1}^{\infty} p(k)E(k,\nu)$. *If the process is observed at an arbitrary time, then the probability that there are* r *stages left to complete is given by*

$$H(r) = \frac{1}{\nu\tau}\left(\sum_{k=r}^{\infty} p(k)\right) \tag{5.2.1}$$

where τ *is the mean time until the next renewal.*

For a proof, see [24]. Notice that as a special case, the probability that there is one lifetime left to complete is given by $1/\nu\tau$.

5.2.2 Probabilistic Transitions

A gPEPA model with generally distributed activities can be interpreted as one with probabilistic transitions, and without general distributions.

Definition 5.2.1. *A function* p approximates X *if and only if for all* $\epsilon > 0, t > 0$, $|F_X(t) - \sum_{j=1}^{\infty} p(j)E(j,\nu)(t)| < \epsilon$, *for some fixed* ν.

Therefore, $p(\cdot)$ approximates X, if with a suitable choice of ν, $p(\cdot)$ defines a probability distribution over a set of Erlang-k distributions such that the distribution function of the mixture is arbitrarily close to F_X. A function with such a property is subscripted with the random variable, for example $p_X(\cdot)$. The rules given in Figure 5.1 define the first stage of a probabilistic semantics for gPEPA. H and J range over multisets of pairs of gPEPA processes and random variables; A and B range over multisets of random variables. Collectively, this set of rules is called **Pr Rules**.

These rules translate a generally distributed behaviour into what will be interpreted as a set of probabilistic transitions. The intuition is that such transitions will be resolved instantaneously, and the process will then behave like a linear PEPA process with a fixed number of lifetimes—the choice of probabilistic transition defines the number of lifetimes chosen. After the linear process completes each lifetime, another probabilistic choice is made, this time to determine the successor derivative.

Let the operational rules for PEPA processes be called **PEPA Rules**. Then the semantics for gPEPA processes are given by insisting that all probabilistic rules be applied first, if possible; that is, given a gPEPA process, if a probabilistic transition can be inferred, then for that process, no conventional exponential transitions can be inferred. This assertion is formalised in Section 5.2.2.1.

Definition 5.2.2. *For a gPEPA process* $\overline{P}$, *let* $G = \{\!|\overline{P} \overset{A,k}{\dashrightarrow} c[(H,k)] : k \in \mathbb{N}|\!\}$ *be the multiset of probabilistic transitions it is possible to infer by* **Pr Rules**, *for some static context* $c[.]$. *Let* $p_G(\cdot)$ *be a function over* $k \in \mathbb{N}$ *such that* $p_G(\cdot)$ *approximates* $\min_{\{\!|X\,:\,(F,X)\in H|\!\}}$. *Then* $p_G(\cdot)$ *is the* probabilistic choice resolving *function for* G *i.e.*

$$\Pr(\overline{P} \overset{A,i}{\dashrightarrow} (H,i) \text{ is chosen}) = p_G(i)$$

Definition 5.2.3. (H,k) *is a multi-stage linear PEPA process such that*

$$(H,k) \stackrel{\text{def}}{=} \begin{cases} (\tau, r).(H, k-1) & \text{if } k > 0, \text{ where } r \text{ is the fixed mean of the Erlang} \\ & \text{mixture which approximates } \min_{\{\!|X\,:\,(F,X)\in H|\!\}} \\ (H,0) & \text{otherwise (for which see Figure 5.2 later)} \end{cases}$$

Pr Prefix states that when a generally distributed activity is enabled, it is possible to infer a probabilistic transition to a k-stage compound process. Notice that from one generally distributed activity, it is possible to infer countably infinitely many probabilistic transitions. This is because the general distribution is modelled by a probabilistic sum over Erlang-k distributions. In fact, the probabilistic choice resolution function (Definition 5.2.2) gives a probability space over all such k; this will give the probability of selecting any particular Erlang-k sequence of activities, so as to accurately mimic the generally distributed lifetime.

Pr Prefix

$$\frac{}{(\alpha, X).F \overset{\{\!|X|\!\},k}{\dashrightarrow} (\{\!|(F, X)|\!\}, k)}$$

for all $k \in \mathbb{N}$

Pr Choice

$$\frac{F \overset{A,k}{\dashrightarrow} (H, k) \qquad G \overset{B,i}{\dashrightarrow} (J, i)}{F + G \overset{A \uplus B,j}{\dashrightarrow} (H \uplus J, j)}$$

for all $j \in \mathbb{N}$

Pr Mixed Choice

$$\frac{F \overset{A,k}{\dashrightarrow} (H, k) \qquad G \xrightarrow{(\alpha,r)} G'}{F + G \overset{A \uplus \{\!|X|\!\},j}{\dashrightarrow} (H \uplus \{\!|(G', X)|\!\}, j)} \quad F_X(t) = 1 - e^{-rt}$$

$$\frac{F \xrightarrow{(\alpha,r)} F' \qquad G \overset{A,k}{\dashrightarrow} (H, k)}{F + G \overset{A \uplus \{\!|X|\!\},j}{\dashrightarrow} (H \uplus \{\!|(F', X)|\!\}, j)} \quad F_X(t) = 1 - e^{-rt}$$

for all $j \in \mathbb{N}$

Pr Hide

$$\frac{F \overset{A,k}{\dashrightarrow} (H, k)}{F/L \overset{A,k}{\dashrightarrow} (H, k)/L}$$

Pr Coop

$$\frac{F \overset{A,k}{\dashrightarrow} (H, k)}{F \underset{L}{\bowtie} G \overset{A,k}{\dashrightarrow} (H, k) \underset{L}{\bowtie} G}$$

$$\frac{G \overset{B,i}{\dashrightarrow} (J, i)}{F \underset{L}{\bowtie} G \overset{B,i}{\dashrightarrow} F \underset{L}{\bowtie} (J, i)}$$

Pr Const

$$\frac{F \overset{A,k}{\dashrightarrow} (H, k)}{P \overset{A,k}{\dashrightarrow} (H, k)} \quad P \overset{\text{def}}{=} F$$

Figure 5.1: Introducing probabilistic transitions

Pr Choice ensures that meaningful probabilities can be inferred when a choice is available over generally distributed activities. Competitive choice raises an important issue. Consider the following gPEPA process:

$$\overline{P} \stackrel{\text{def}}{=} (\alpha, X).\overline{Q} + (\beta, Y).\overline{R}$$
$$\text{where } F_X = \mathsf{Uni}(1,3), F_Y = \mathsf{Uni}(2,4) \qquad (5.2.2)$$

Both enabled activities are generally distributed. However, which activity completes first (and thus disables the other) is dependent on how much time has elapsed since both activities were enabled. If strictly between 1 and 2 units of time have passed, the second activity cannot possibly have completed. Therefore, it should not be possible to infer this transition, during this time period. In the terminology of Rumsewicz and Henderson [65], $\overline{P}$ exhibits *age-dependent routing*.

The approach taken by rule **Pr Choice** is to merge the distributions, in the sense that the random variables associated with the activities enabled by each side of the choice are pooled together. The probabilistic choice resolution function ensures that the weighted sum over Erlang-k distributions will give an expected duration equal to the expectation of the minimum of the pooled random variables. When a branch is chosen, and each lifetime has run down, the probability that one possible derivative will be the choice of successor is given by the *time-averaged* probability that the random variable of the activity which leads to it is less than every other candidate random variable. This means the approach taken is *time-independent*; the probability of one derivative resulting does not change as time passes. It is certainly not clear that this is consistent with intuition—for example, consider the intuitive time-dependent behaviour of the gPEPA process shown in Equation (5.2.2). However in the cases examined in this thesis, the correctness of this translation can be shown by making use of the following result due to Rumsewicz and Henderson [65]:

Theorem 5.2.1. (Rumsewicz and Henderson) *Let $\overline{P}$ be a (stochastic) process with age-dependent routing probabilities. Let Q be process $\overline{P}$ except with time-averaged age-independent routing probabilities. If Q is insensitive to its generally distributed transitions, then the equilibrium distribution of $\overline{P}$ is equivalent to that of Q.*

This means that when it can be shown that the time-averaged process underlying a gPEPA model is insensitive to its generally distributed transitions, the distribution of this process is provably equivalent to that for the original time-dependent process, and thus there is no loss of accuracy in the solution provided. The result presented in this chapter will demonstrate partial balance conditions guaranteeing insensitivity of residence time in particular components, and of the enabled generally distributed activities, allowing Theorem 5.2.1 to be applied. **Pr Mixed Choice** ensures that when a choice context enables both a generally distributed and an exponentially distributed lifetime, that the exponential random variable is subsumed, and the competition is treated in the same way as for **Pr Choice**.

Rules **Pr Coop** and **Pr Hide** allow probabilistic transitions to be inferred within static contexts. Notice that there is no rule allowing any kind of synchronisation on a probabilistic transition—this is consistent with the restriction that processes may not cooperate on generally distributed activities. Problems arise if both subcomponents of a PEPA cooperation can perform a probabilistic transition. However this is forbidden by the assumption that no two generally distributed activities may be enabled by processes in parallel at the same time. Recall that this is allowed in a choice context, in which case the outcome is captured by merging the competitors into one generally distributed lifetime, and considering the expected duration as the minimum of the durations of the competitors.

The final task is to define the second stage of the probabilistic semantics for gPEPA. This consists of a set of rules, presented in Figure 5.2, which allows the inference of a successor derivative once each lifetime has completed. Collectively, these are known as **Successor Rules**. These rules induce a new probabilistic transition relation, this time

Successor

$$\frac{}{(H, 0) \overset{k}{\dashrightarrow} F_i}$$

$$\text{if } H = \{\!|(F_1, X_1), \ldots, (F_n, X_n)|\!\} \\ \text{and } \Pr(X_i < \min_{\{\!| X_j \,:\, (F_j, X_j) \in H, j \neq i |\!\}}) = k$$

Succ Hide

$$\frac{(H, 0) \overset{k}{\dashrightarrow} F}{(H, 0)/L \overset{k}{\dashrightarrow} F/L}$$

Succ Coop

$$\frac{(H, 0) \overset{k}{\dashrightarrow} F}{(H, 0) \underset{L}{\bowtie} G \overset{k}{\dashrightarrow} F \underset{L}{\bowtie} G} \qquad \frac{(H, 0) \overset{k}{\dashrightarrow} G}{F \underset{L}{\bowtie} (H, 0) \overset{k}{\dashrightarrow} F \underset{L}{\bowtie} G}$$

Figure 5.2: Choosing a successor derivative

where the transition is decorated with a value $k \in [0, 1]$. This value k is to be treated as the probability this transition will be chosen. That a probabilistic interpretation is valid is stated in the following lemma.

Lemma 5.2.2. *Let $H = \{\!|(F_1, X_1), \ldots, (F_n, X_n)|\!\}$. Then $\sum_{\{\!|(H,0) \overset{k}{\dashrightarrow} F_i \in H|\!\}} k = 1$.*

Proof. Omitted. $\square$

Successor chooses a successor from $H = \{\!|(F_1, X_1), \ldots, (F_n, X_n)|\!\}$, a multiset of possibilities. This is done on the basis of the random variables originally associated with activities leading to these successors. Lemma 5.2.2 shows that the sum over the labels of all possible transitions from a given $(H, 0)$ is equal to 1. The expression giving each transition label k to derivative F_i is the probability that X_i will complete before any other X_j, $1 \leq j \leq n, j \neq i$, in an n-way race.

Rules **Succ Hide** and **Succ Coop** ensure that the probabilistic behaviour of a process $(H, 0)$ can be inferred from within a static context. Note there is no such rule for a choice context—this is because it is impossible to have a process of the form $(H, 0) + \overline{Q}$. The following lemma proves this fact.

Lemma 5.2.3. *Let $\overline{P}$ be a gPEPA process. Then by making transitions from $\overline{P}$, it is not possible to reach a process of the form $(H, 0) + \overline{Q}$ for any H, $\overline{Q}$.*

Proof. The proof is by induction on the structure of $\overline{P}$. Of interest in this proof is that it is necessary to consider processes built using generally distributed activity prefix.

Case $(\alpha, r).\overline{P'}$:

By **Prefix**, $(\alpha, r).\overline{P'} \xrightarrow{(\alpha,r)} \overline{P'}$. By the inductive hypothesis, $\overline{P'}$ cannot reach $(H, 0) + \overline{Q}$, and so neither can $\overline{P}$.

Case $(\alpha, X).\overline{P'}$:

By **Pr Prefix**, $(\alpha, X).\overline{P'} \overset{\{\!|X|\!\},k}{\dashrightarrow} (\{\!|(\overline{P'}, X)|\!\}, k)$ for all $k \in \mathbb{N}$. By definition, $(\{\!|(\overline{P'}, X)|\!\}, k)$ is equivalent to

$$\overbrace{(\tau, \nu). \ldots .(\tau, \nu).}^{k \text{ times}} (\{\!|(\overline{P'}, X)|\!\}, 0)$$

By k applications of **Prefix**, the result is $(\{\!|(\overline{P'}, X)|\!\}, 0)$. By application of **Successor**, the result is $\overline{P'}$. By the inductive hypothesis, $\overline{P'}$ cannot reach $(H, 0) + \overline{Q}$, and so neither can $\overline{P}$.

Case $\overline{R} + \overline{S}$:

One of three inference rules may be used in this case—**Choice**, **Pr Choice** or **Pr Mixed Choice**.

Case Choice :

This rule is only applicable if neither $\overline{R}$ nor $\overline{S}$ enable probabilistic transitions.

By **Choice**, $\overline{P} \xrightarrow{(\alpha,r)} \overline{P'}$ if either $\overline{R} \xrightarrow{(\alpha,r)} \overline{P'}$ or $\overline{S} \xrightarrow{(\alpha,r)} \overline{P'}$. Assume the former (the latter case is completely analogous). By the inductive hypothesis, $\overline{R}$ cannot reach $(H, 0) + \overline{Q}$ and so $\overline{P'}$ cannot. Since $\overline{P} \xrightarrow{(\alpha,r)} \overline{P'}$, $\overline{P}$ cannot reach $(H, 0) + \overline{Q}$.

Case Pr Choice :

$\overline{P} \overset{A \uplus B, j}{\dashrightarrow} (L \uplus J, j)$ if and only if both $\overline{R} \overset{B,i}{\dashrightarrow} (L, i)$ and $\overline{S} \overset{A,k}{\dashrightarrow} (J, k)$. In similar style to the **Pr Prefix** case, $\overline{R}$ may evolve into some $\overline{R'}$ such that $(\overline{R'}, X) \in L$, and similarly for $\overline{S}$ and $\overline{S'}$. By the inductive hypothesis, neither $\overline{R}$ nor $\overline{S}$ may reach $(H, 0) + \overline{Q}$; the multiset of derivatives reachable by $\overline{P}$ is equal to $L \uplus J$, and so $\overline{P}$ cannot reach $(H, 0) + \overline{Q}$ either.

Case Pr Mixed Choice :

By a similar argument.

Case $\overline{R} \underset{L}{\bowtie} \overline{S}$:

Both sets of inference rules, **Coop** and **Pr Coop**, lead to derivatives which retain the static structure of $\overline{P}$; therefore no derivative may be of the form $(H, 0) + \overline{Q}$ (exhibiting a dynamic choice context).

Case $\overline{R}/L$:

By a similar argument to the case $\overline{R} \underset{L}{\bowtie} \overline{S}$.

Case $\overline{P} \stackrel{\text{def}}{=} \overline{R}$:

By either rule **Const** or **Pr Const**, the behaviour of $\overline{P}$ is identical to that of $\overline{R}$. $\overline{R}$ is guaranteed not to reach $(H, 0) + \overline{Q}$ by the cases above, and so the same is true for $\overline{P}$.

□

It will be helpful to understand this system of rules with a worked example. In this example, for the sake of analytic simplicity, the random variables used are assumed to be exponentially distributed.

Example 5. *Consider the following gPEPA model:*

$$\begin{aligned}
\overline{P} &\stackrel{\text{def}}{=} (\alpha, X).\overline{P'} \\
\overline{Q} &\stackrel{\text{def}}{=} (\beta, Y).\overline{Q'} \\
\overline{R} &\stackrel{\text{def}}{=} (\gamma, r).\overline{R'} \\
\overline{S} &\stackrel{\text{def}}{=} (\overline{P} + \overline{Q}) \underset{L}{\bowtie} \overline{R} \\
&\quad \textit{where } \alpha, \beta, \gamma \notin L, F_X(t) = 1 - e^{-\lambda t}, F_Y(t) = 1 - e^{-\mu t}
\end{aligned} \tag{5.2.3}$$

Now the rules are applied to derive the possible transitions for $\overline{S}$.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\alpha, X).\overline{P'} \overset{\{\!|X|\!\},j}{\dashrightarrow} (\{\!|(\overline{P'}, X)|\!\}, j)}\,(\mathit{by}\ \mathbf{PrPrefix})}{\overline{P} \overset{\{\!|X|\!\},j}{\dashrightarrow} (\{\!|(\overline{P'}, X)|\!\}, j)}\,(\mathit{by}\ \mathbf{PrConst}) \qquad \dfrac{\vdots}{\overline{Q} \overset{\{\!|Y|\!\},i}{\dashrightarrow} (\{\!|(\overline{Q'}, Y)|\!\}, i)}}{\overline{P} + \overline{Q} \overset{\{\!|X,Y|\!\},k}{\dashrightarrow} (\{\!|(\overline{P'}, X), (\overline{Q'}, Y)|\!\}, k)}\,(\mathit{by}\ \mathbf{PrChoice})}{(\overline{P} + \overline{Q}) \underset{L}{\bowtie} \overline{R} \overset{\{\!|X,Y|\!\},k}{\dashrightarrow} (\{\!|(\overline{P'}, X), (\overline{Q'}, Y)|\!\}, k) \underset{L}{\bowtie} \overline{R}}\,(\mathit{by}\ \mathbf{PrCoop})}{\overline{S} \overset{\{\!|X,Y|\!\},k}{\dashrightarrow} (\{\!|(\overline{P'}, X), (\overline{Q'}, Y)|\!\}, k) \underset{L}{\bowtie} \overline{R}}\,(\mathit{by}\ \mathbf{PrConst})
$$

Note that any *k may be inferred. The behaviour of $(\{\!|(\overline{P'}, X), (\overline{Q'}, Y)|\!\}, k)$ is given by Definition 5.2.3; it will perform k exponentially distributed τ activities, each with a rate r identical to that present in the Erlang mixture distribution which approximates the distribution function of* $\min_{\{\!|X,Y|\!\}}$.

$F_{\min_{\{\!|X,Y|\!\}}} = 1 - e^{-(\lambda+\mu)t}$*, so a degenerate mixture is suitable in this case, containing one branch to an Erlang-1 distribution with parameter $\lambda + \mu$. By definition, the transitions inferred are actually*

$$
\overline{S} \overset{\{\!|X,Y|\!\},k}{\dashrightarrow} \overbrace{(\tau, \lambda + \mu). \ldots .(\tau, \lambda + \mu)}^{k\ \mathit{times}}.(\{\!|(\overline{P'}, X), (\overline{Q'}, Y)|\!\}, 0) \underset{L}{\bowtie} \overline{R} \quad \mathit{for\ all}\ k
$$

However, the probabilistic choice resolving function for this mixture, $p_{\{\!|X,Y|\!\}}(\cdot)$ is such that

$$
p_{\{\!|X,Y|\!\}}(i) = \begin{cases} 1 & \mathit{if}\ i = 1 \\ 0 & \mathit{otherwise} \end{cases}
$$

Therefore the only transition possible in this case is

$$
\overline{S} \overset{\{\!|X,Y|\!\},1}{\dashrightarrow} (\tau, \lambda + \mu).(\{\!|(\overline{P'}, X), (\overline{Q'}, Y)|\!\}, 0) \underset{L}{\bowtie} \overline{R}
$$

It is impossible to now infer a probabilistic transition, and so using **PEPA Rules** *as usual, the τ transition can be inferred. Now finally, the rules of Figure 5.2 are employed to give the following.*

$$
\dfrac{\dfrac{}{(\{\!|(\overline{P'}, X), (\overline{Q'}, Y)|\!\}, 0) \overset{\lambda/\lambda+\mu}{\dashrightarrow} \overline{P'}}\,(\mathit{by}\ \mathbf{Successor})}{(\{\!|(\overline{P'}, X), (\overline{Q'}, Y)|\!\}, 0) \underset{L}{\bowtie} \overline{R} \overset{\lambda/\lambda+\mu}{\dashrightarrow} \overline{P'} \underset{L}{\bowtie} \overline{R}}\,(\mathit{by}\ \mathbf{Succ\ Coop})
$$

In this example with exponential distributions, it is clear the semantics will be in tune with intuition—the expected time to resolve the choice will be $1/(\lambda + \mu)$, and the prob-

ability that $\overline{P'}$ will result is $\lambda/(\lambda+\mu)$, since:

$$\begin{aligned}
&\int_0^\infty (1 - F_Y(t))dF_X(t) \\
&= \int_0^\infty e^{-\mu t} f_X(t)dt \\
&= \int_0^\infty e^{-\mu t} \lambda e^{-\lambda t} dt \\
&= \left[\lambda/(\lambda+\mu)e^{-(\lambda+\mu)t}\right]_0^\infty \\
&= \lambda/(\lambda+\mu)
\end{aligned} \tag{5.2.4}$$

Let **gPEPA Rules** = **Pr Rules** $\cup$ **Successor Rules**. By use of **gPEPA Rules**, the behaviour of every gPEPA model with generally distributed activities may be understood in terms of probabilistic transitions and exponentially distributed activities only.

5.2.2.1 Consistency of the Probabilistic Rules

It was stated informally in Section 5.2.2 that in order to derive a transition system for a gPEPA model, the rules for probabilistic transitions *must be applied first*, such that a process could not enable any exponential transitions if it enabled probabilistic transitions. This simplified the presentation, but more formally, the transition system specification should contain a modified version of each rule in **PEPA Rules**. These modified rules require the addition of *negative premises*. For example, consider the rule for exponential choice given in Figure 5.3. Such an exponential transition may only

$$\textbf{Mod Choice} \quad \frac{E \xrightarrow{(\alpha,r)} E' \quad E \not\dashrightarrow \quad F \not\dashrightarrow}{E + F \xrightarrow{(\alpha,r)} E'}$$

Figure 5.3: PEPA Choice rule with negative premises

be inferred if neither subprocess is capable of enabling probabilistic transitions. The new set of rules **Mod PEPA Rules** contains modified versions of each rule in **PEPA Rules** *except* **Prefix**; this remains an axiom, since the only capability a prefix process has is of performing an exponential activity.

With negative premises present in the operational rules, it is by no means clear that the rules are consistent, and that a transition system may be derived. Such problems were described by Groote [31]. To illustrate, Groote presents the following operational rule (amongst several others).

$$\frac{F \not\longrightarrow F}{F \longrightarrow F}$$

Of course, no such transition system is consistent with this rule. Given a set of rules, Groote constructs a *literal dependency graph*, where

- a p-labelled edge exists from ψ to ϕ if ψ is a positive premise of a rule where ϕ is the conclusion (allowing substitutions for variables in rules);
- a n-labelled edge exists from ψ to ϕ if ψ is a negative premise of a rule where ϕ is the conclusion.

Then a cycle in the dependency graph with a negative edge indicates a contradiction, that a transition may be inferred if and only if it may not. **gPEPA Rules** do lead to a consistent transition system. This fact is proven in the following lemma.

Lemma 5.2.4. *Let $\mathcal{R} = \longrightarrow \cup \dashrightarrow$. Then for no gPEPA processes $\overline{P}, \overline{Q}$ is it the case that $(\overline{P}, \overline{Q}) \in \mathcal{R}$ if and only if $(\overline{P}, \overline{Q}) \notin \mathcal{R}$.*

Proof. The argument presented here is informal, but illustrates that there can be no such cycle with a negative edge in a literal dependency graph derived from **gPEPA Rules**. Consider the subsets of rules that make up **gPEPA Rules**. The only rules which contain negative premises are contained in **Mod PEPA Rules**. By analogy with the rule presented in Figure 5.3, each of these rules implies an exponential transition if it is possible to refute one or more probabilistic transitions. Only two rules contained in **Pr Rules** conclude in a probabilistic transition with an exponential transition premise—these are both labelled **Pr Mixed Choice**. Therefore the only potential for a negative cycle derives from an inference tree of the following form:

$$\dfrac{\dfrac{\vdots}{\overline{P} \overset{A,k}{\dashrightarrow} (H,k)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\cdots \quad \overline{P}+\overline{Q} \longrightarrow \cdots \qquad \overline{P}+\overline{Q} \not\dashrightarrow}{\vdots}\ (by\ \textbf{Const})}{\overline{R} \longrightarrow \cdots}}{\vdots}}{\overline{Q} \xrightarrow{(\alpha,r)} \overline{Q'}}\ (by\ \textbf{PrCoop})}{\overline{P}+\overline{Q} \overset{A \uplus \{\!|X|\!\},j}{\dashrightarrow} (H \uplus \{\!|(\overline{P'},X)|\!\}, j)}\ F_X(t) = 1 - e^{-rt}$$

The reasoning is as follows. It must be the case that an application of **Const** is present because this is the only PEPA rule which implies a transition from a process simpler in structure than that appearing in the premise. From the transition of $\overline{R}$, it must then be possible to infer the exponential transition of $\overline{Q}$. However, this tree implies that a gPEPA process of the form $\overline{R} \stackrel{\text{def}}{=} c[\overline{P} + \overline{Q}]$ exists, and also that $\overline{Q} \stackrel{\text{def}}{=} c'[\overline{R}]$, for some contexts c, c' (or if not $\overline{R}$, then some constant $\overline{S}$ defined to depend upon $\overline{R}$). This is explicitly forbidden in PEPA, and thus in gPEPA; it must be possible to unwind each process specification until all variables are replaced by guarded terms. Therefore, no inference tree of the form above may exist. $\square$

gPEPA Rules lead to the following *derivation graph* model for a gPEPA process, with no reference to generally distributed sojourn times in states. The model is given by $(\mathcal{C}, \mathcal{A}ct, \longrightarrow, \dashrightarrow)$, where

$$\dashrightarrow \subseteq (\mathcal{C} \times (2^{PRV} \times \mathbb{N}) \times \mathcal{C}) \cup (\mathcal{C} \times [0,1] \times \mathcal{C})$$

and

- $\longrightarrow$ is the least relation which can be inferred under **Mod PEPA Rules**, and
- $\dashrightarrow$ is the least relation which can be inferred under **gPEPA Rules**.

Notice that such models are potentially infinite-state. However if such a model is built by application of the presented transition rules, this extra data is used only in a very restricted way.

5.2.3 Interpreting a gPEPA Model as a Stochastic Process

In this section, it is shown how the derivation graph of a gPEPA process can be used to generate a representation of the model as a stochastic process. It is shown that by applying the probabilistic semantics, and with an intuitive operational interpretation of the probabilistic transitions, the resulting stochastic process is a potentially infinite-state continuous time Markov chain.

A naïve approach is taken to generate the stochastic process. Assume a gPEPA process $\overline{P}$ with a model of the form $(\mathcal{C}, \mathcal{A}ct, \longrightarrow, \dashrightarrow)$. This defines a set of states which is infinite with the introduction of any generally distributed activity, and two transition relations over the state space. The first is $\longrightarrow$ which is a set of exponentially distributed transitions, and the second is $\dashrightarrow$ which is a set of probabilistic transitions. In order to form the stochastic process, a state is associated with each node of the graph, and the transitions between states are defined by use of a combination of both model transition relations.

On entering a state in which probabilistic transitions are enabled, the choice of which one to take is deemed to be resolved instantaneously; if exponential transitions are enabled, they are deemed to race, and the successor state is determined by the transition which completes first. It will be proved that no sequential component enables both probabilistic and exponential transitions. This leads to a demonstration that the resulting process is a continuous time Markov chain.

A derivative of the process $\overline{P}$ may be given the representation $(\overline{S_1}, \ldots, \overline{S_n})_{\overline{P}}$, where each $\overline{S_i}$ is a sequential component. The subscript $\overline{P}$ is required since knowledge of the static structure of P must be retained in order to reason about the process's behaviour. However, this will be omitted when it is clear from context. If the ith sequential component $\overline{S_i}$ currently enables probabilistic activities, it will be denoted by $\widehat{S}_i$; otherwise, it will be denoted by $\overline{S_i}$. The desired behaviour of a gPEPA model is that when a set of probabilistic transitions are enabled, the choice is resolved instantaneously. Therefore, the stochastic process will witness no time passing in any state in which probabilistic transitions are enabled. For this reason, such states are excluded from the state space of the stochastic process. The states of the stochastic process are given by $\{\overline{Q} \in ds(\overline{P}) : \overline{Q} \equiv (\overline{S_{i_1}}, \ldots, \overline{S_{i_n}})_{\overline{P}}\}$.

Lemma 5.2.5. *A sequential component cannot enable both probabilistic and exponential transitions.*

Proof. Consider any sequential component $\overline{S}$ such that $\overline{S} \overset{A,k}{\dashrightarrow} (K, k)$. The proof that $\overline{S}$ does not enable any exponential transitions is by structural induction on the depth of inference of $\overline{S} \overset{A,k}{\dashrightarrow} (K, k)$.

Case $\overline{S} \equiv \overline{T} + \overline{V}$:

If $\overline{S} \overset{A,k}{\dashrightarrow} (K, k)$ then its behaviour may be inferred by one of two rules.

Case Pr Choice :

Then both $\overline{T} \overset{B,i}{\dashrightarrow} (H, i)$ and $\overline{V} \overset{C,j}{\dashrightarrow} (J, j)$ where $A = B \uplus C$ and $K = H \uplus J$. By the inductive hypothesis, both $\overline{T} \not\longrightarrow$ and $\overline{V} \not\longrightarrow$, and therefore it is impossible to infer that $S \equiv T + V \longrightarrow$.

Case Pr Mixed Choice :

Then without loss of generality, $\overline{T} \overset{B,i}{\dashrightarrow} (H, i)$ and $\overline{V} \xrightarrow{(\alpha,r)} \overline{V'}$ where $F_X(t) = 1 - e^{-rt}$, $A = B \uplus \{\!|X|\!\}$ and $K = H \uplus \{\!|(\overline{V'}, X)|\!\}$. By the inductive hypothesis, $\overline{T} \not\longrightarrow$ and $\overline{V} \not\dashrightarrow$. The only possibility to infer $\overline{S} \longrightarrow$ is through **Mod Choice** but this rule insists that if $\overline{V} \longrightarrow$ then $\overline{T} \not\dashrightarrow$; however this contradicts the inference that $\overline{T} \overset{B,i}{\dashrightarrow} (H, i)$.

Case $\overline{S} \equiv (\alpha, X).\overline{T}$:

There is no rule which allows the inference of an exponential transition from a process of the form $(\alpha, X).\overline{T}$.

Case $\overline{S} \equiv (\alpha, r).\overline{T}$:

There is no rule which allows the inference of a probabilistic transition from a process of the form $(\alpha, r).\overline{T}$, so this case is true trivially.

Secondly, consider any component $\overline{P}$ such that $\overline{P} \overset{k}{\dashrightarrow} \overline{P'}$. By inspection of **Successor Rules**, it can be seen that $\overline{P} \equiv c[(H, 0)]$, for some H and *static* context $c[.]$. Therefore if $\overline{P} \equiv \overline{S}$ is sequential, the only inference rule available is **Successor**; therefore there is no potential for the inference of exponential transitions. $\square$

Theorem 5.2.2. *Let* $\overline{P} \equiv P_0$ *such that* $P_i, i \in \mathbb{N}$ *is a (possibly countably infinite) enumeration of* $ds(\overline{P})\backslash\{P' : P' \dashrightarrow\}$. *Define the stochastic process* $\{X_t : t \in \mathbb{R}^+\}$ *such that* $X_t = P_i$ *indicates that the model behaves as component* P_i *at time* t. *Then* $\{X_t\}$ *is a continuous time Markov chain.*

Proof. $\{X_t\}$ has the Markov property if and only if for $t_0 < t_1 < \ldots < t_n < t_{n+1}$, the joint probability distribution of $(X_{t_0}, X_{t_1}, \ldots, X_{t_n}, X_{t_{n+1}})$ is such that

$$\Pr(X_{t_{n+1}} = P_{i_{n+1}} : X_{t_0} = P_{i_0}, \ldots, X_{t_n} = P_{i_n}) = \Pr(X_{t_{n+1}} = P_{i_{n+1}} : X_{t_n} = P_{i_n})$$

This property means that given the state of the process at the current time, the future behaviour, conditional on the current behaviour, is independent of all past behaviour. This is equivalent to the property that the distribution of the time until the next state change is independent of the time already spent in the current state.

Let $X_t = \overline{Q} \equiv (S_{i_0}, \ldots, S_{i_n})_{\overline{P}}$. Consider Y_i, a random variable representing the sojourn time in $\overline{Q}$. This state will enable a set of exponentially distributed activities, some of which might be the result of a cooperation between (possibly several) sequential components. The sojourn time distribution, conditional on the completion of a particular activity $\mathtt{a}$, is given by $S_{i\mathtt{a}}(t) = \Pr(Y_i \leq t \mid \mathtt{a} \text{ completes})$. Therefore, the unconditional sojourn time is given by

$$S_i(t) = \sum_{\mathtt{a} \in ds(\overline{Q})} \Pr(Y_i \leq t \mid \mathtt{a} \text{ completes}) = \sum_{\mathtt{a} \in ds(\overline{Q})} S_{i\mathtt{a}}(t)$$

As Hillston shows in [44], this sojourn time is exponentially distributed, with a rate equal to the sum of the rates of the individual enabled activities. Denote the successor state $\overline{Q'} \equiv (\overline{S_{j_1}}, \ldots, \overline{S_{j_n}})_{\overline{P}}$. Note that $\overline{Q'}$ does not necessarily meet the requirements for being a state of the stochastic process—some of the sequential components may enable probabilistic transitions. However, by Lemma 5.2.5, each sequential component enables either probabilistic transitions, or exponentially distributed activities, but not both. For each sequential component $\widehat{S_i}$ enabling probabilistic transitions, a successor derivative is deemed to be chosen accordingly and instantaneously. The order chosen is irrelevant, since the probabilistic transitions of one sequential component cannot interact with those of any other component. Therefore, without further passing of time, the process evolves to a state $\overline{Q''} \equiv (\overline{S_{k_1}}, \ldots, \overline{S_{k_n}})_{\overline{P}}$, where for each $\overline{S_{k_i}}$, $\overline{S_{k_i}} \not\dashrightarrow$. Therefore this derivative is a state of the stochastic process. This shows that the distribution of time until the next state change of the stochastic process is exponentially distributed, and hence the stochastic process has the Markov property. $\square$

One issue is in which state the stochastic process $\overline{P}$ begins. If any of the $\overline{S_{0_i}}$ enable generally distributed activities, then immediately $\overline{P}$ enables a set of probabilistic transitions, and strictly, $\overline{P}$ is not a state of the stochastic process. A solution is to instantaneously resolve all probabilistic choices, leading to a state which is a state of the stochastic process. It might seem that this influences the future behaviour of the process; however this is an academic point, since firstly, the generally distributed activities will introduce no deadlocks and will not affect the positive recurrence property of the stochastic process (the states of the process representing the stages of a generally distributed activity will be positive recurrent), and secondly, the interest in this chapter is in steady-state only. In the rest of this chapter, a sequential component which enables only exponentially distributed activities will be referred to as *exponential*. If a sequential component enables any generally distributed activities, it will be referred to as *generalised*.

5.3 Balance Equations for General Distributions

This section examines the conditions under which a gPEPA process has an equilibrium solution identical to that which it would have with exponentially distributed activities only. These conditions take the form of partial balance equations over the state space of the stochastic process underlying a gPEPA model. First it is established when a gPEPA process has an equilibrium solution.

Definition 5.3.1 (Cyclic gPEPA process). *A gPEPA process is* cyclic*, or* irreducible*, if $\overline{P} \in ds(\overline{P'})$ for all $\overline{P'} \in ds(\overline{P})$.*

By the same reasoning discussed in Section 2.3.4, it is the case that for a gPEPA component to be irreducible, it must not feature static combinators, that is hiding or cooperation, within the context of a choice combinator. Furthermore, the continuous time Markov chain underlying a gPEPA process is irreducible if and only if the process is cyclic. As a consequence, each sequential component must also be cyclic. It is assumed the stochastic process underlying a gPEPA model is time homogeneous since no features of a gPEPA model depend on time.

Lemma 5.3.1. *Let $\overline{P}$ be a cyclic gPEPA process. Let $\{X_t\}$ be the stochastic process underlying $\overline{P}$ such that the sample space of each X_t is given by $ds(\overline{P})$. Let $T_{\overline{P}}$ be a random variable representing the time at which $\{X_t\}$ returns to $\overline{P}$ after starting in $\overline{P}$ (independent of time). Then for each $\overline{P'} \in ds(\overline{P})$, $E[T_{\overline{P'}}] < \infty$.*

Proof.

Let $\overline{P'} \in ds(\overline{P})$. Without loss of generality, $\overline{P}$ can be represented as $(\overline{S_1}, \ldots, \overline{S_n})_{\overline{P}}$. The subset of states $\{\overline{P'} \in ds(\overline{P}) : \overline{P'} \not\dashrightarrow\}$ is finite, for the same reason that the derivation graph of a PEPA process is finite. From this set, construct the *reduced derivation graph* such that $\overline{P} \Longrightarrow \overline{P'}$ if and only if either $\overline{P} \xrightarrow{(\alpha,r)} \overline{P'}$ or $\overline{P} \dashrightarrow \longrightarrow^k \dashrightarrow \overline{P'}$ for any finite k by the evolution of a particular $\overline{S_i}$ only. There is at least one finite path from $\overline{P'}$ to $\overline{P}$ in the reduced derivation graph. It is shown (informally) by induction on the length of such a path that the expected time to return is finite.

Base Case :

Let $\overline{P'} \Longrightarrow \overline{P}$. If $\overline{P'} \xrightarrow{(\alpha,r)} \overline{P}$ then the expected time to reach $\overline{P}$ is $1/r$. Otherwise, $\overline{P'}$ reaches $\overline{P}$ because one of its sequential components $\overline{S_i}$ performs a generally distributed activity, some (β, X). The probabilistic model ensures that the expected time to perform this activity is equal to $E[X]$. Therefore, the base case holds.

Inductive Case :

This argument is simpler. Let $\overline{P} \Longrightarrow \overline{P''} \Longrightarrow^k \overline{P}$. Then by the same argument above, the expected time to reach $\overline{P''}$ is finite; and by the inductive hypothesis, the expected time to reach $\overline{P}$ from $\overline{P''}$ is finite.

$\square$

It is henceforth assumed that the stochastic process underlying any gPEPA model $\overline{P}$ has a steady-state solution (that is, that any gPEPA model is cyclic).

Now, some notation used in describing a set of balance equation conditions is introduced. When referring to a component, or derivative, in the context of the gPEPA model, or its transition system, $\overline{P}$ or $\overline{Q}$ is used; when referring to either a PEPA or gPEPA model, P or Q is used. Since the state space of the underlying stochastic process consists of gPEPA derivatives, the states of the stochastic process are denoted $\overline{P}$ and $\overline{Q}$. Recall that the state space of the stochastic process underlying $\overline{P}$ is given by $ds(\overline{P}) \backslash \{\overline{P'} : \overline{P'} \dashrightarrow\}$. The probability of being in a particular state P of the process at steady-state is denoted by $\pi(P)$. In the expression of the balance equations, it will be useful to represent the transition behaviour of a process due to behaviour by the ith sequential component only. The following definition makes this precise.

Definition 5.3.2. *Given $P \equiv (S_0, \ldots, S_n)$, $T \subseteq \{1, \ldots, n\}$, and a binary relation $\mathcal{R}$, $P \mathcal{R}_T P'$ if and only if $P \mathcal{R} P'$ can be inferred using only the sequential components S_i where $i \in T$.*

The derivative set of a process P is the set of components which are reachable from P, or put another way, the least set of processes containing P closed under the transitive closure of the (union of the) transition relation(s). It is convenient to introduce notation to represent the components reachable via one step of a relation.

Definition 5.3.3. $ds_T^{\mathcal{R}}(P) = \{P' \in ds(P) : P \mathcal{R}_T P'\}$.

In the balance equations that follow, this notation is annotated to denote the particular relation $\mathcal{R}$ being employed. Thus, $ds_T^{\longrightarrow}(\overline{P})$ is the set of *all* one-$\longrightarrow$-step derivatives of $\overline{P}$ which can be reached involving only sequential components $\overline{S_i}$ where $i \in T$. If $T = \{1, \ldots, n\}$ then the T may be omitted. The following definition introduces the set of previous derivatives of a component $\overline{P}$; these are simply those derivatives which may make a (one-step) transition to $\overline{P}$.

Definition 5.3.4. $ps_T^{\mathcal{R}}(P) = \{P' : P \in ds_T^{\mathcal{R}}(P')\}$.

When expressing balance equations, it is necessary to talk about the *probabilistic flux* into and out of states of a stochastic process. The sojourn time of residence in a component P is given by a random variable. In the case of a PEPA process, the random variable is exponentially distributed, with a parameter equal to the sum of the rates of the activities it enables. The rate at which the process leaves a component P is denoted by $q(P) = \sum \{\!| \, r : P \xrightarrow{(\alpha,r)} \, |\!\}$. The rate at which component P is left, given that the successor is Q, is denoted by $q(P, Q) = \sum \{\!| \, r : P \xrightarrow{(\alpha,r)} Q \, |\!\}$. This can be further generalised by considering the part played by particular sequential components only.

Definition 5.3.5. *$q(P, P', T)$ represents the instantaneous rate of transition from process P to all P' by the action of components $S_i, i \in T$ and no others. The singleton case may be expressed more concisely as $q(P, P', i)$.*

It is now possible to express concisely the global balance equations for a PEPA process P. The semantics of PEPA ensure that if Q and R are processes, then so is $Q \underset{L}{\bowtie} R$, a cooperation between the two. Therefore, this composed process may cooperate with another component, if the model context allows it. A term expressing each possible combination of cooperations between sequential components S_1 to S_n is contained in

Equation (5.3.1). The characteristic of a process in steady-state is that the probabilistic flux out of any state equals the probabilistic flux into that state. Therefore, the global balance equations for each component P may be expressed as

$$\pi(P)\sum_{R\in 2^S}\sum_{P\in ds_{\overrightarrow{R}}(P)} q(P,P,R) = \sum_{R\in 2^S}\sum_{P\in ps_{\overrightarrow{R}}(P)} \pi(P)q(P,P,R) \qquad (5.3.1)$$
$$\text{where } S=\{1,\ldots,n\}$$

However, a stricter balance may be observed. If the flux into a sequential component S_i, is equal to the flux out of S_i, the component has *sequential local balance.*

Definition 5.3.6. *A sequential component S_i of a PEPA component P, has* sequential local balance *if the flux into any state P due to an activity which leads to the appearance of a sequential component S_i is balanced by the flux out of P due to the completion of an activity enabled by S_i.*

If S_i, a sequential component of P, has sequential local balance, the following equation holds:

$$\pi(P)\sum_{R\in 2^S}\sum_{P'\in ds_{\overrightarrow{R\cup\{i\}}}(P)} q(P,P',R\cup\{i\}) = \sum_{R\in 2^S}\sum_{P'\in ps_{\overrightarrow{R\cup\{i\}}}(P)} \pi(P')q(P',P,R\cup\{i\})$$
$$\text{where } S=\{1,\ldots,n\}\backslash\{i\} \qquad (5.3.2)$$

Recall that the following restrictions are placed on the use of generally distributed activities:

- a cooperation between n components may result in the appearance of no more than one generally distributed activity.
- generally distributed activities may not synchronise (the processes enabling them may not cooperate on these activities).

This means that in effect, sequential components S_i of P which enable generally distributed activities, will have sequential local balance if treating these activities as being exponentially distributed, the following equation holds:

$$\pi(P)\sum_{P'\in ds_{\overrightarrow{i}}(P)} q(P,P',i) = \sum_{R\in 2^S}\sum_{P'\in ps_{\overrightarrow{R\cup\{i\}}}(P)} \pi(P')q(P',P,R\cup\{i\}) \qquad (5.3.3)$$
$$\text{where } S=\{1,\ldots,n\}\backslash\{i\}$$

In the case of a gPEPA process, the underlying transition system also enables probabilistic transitions. Although these contribute nothing to the sojourn time in a state of the stochastic process, they contribute to the probabilistic flux, in that they contribute to the choice of a successor state.

Definition 5.3.7. *$Pr(\overline{P}, \overline{P'}, i)$ represents the probability that the ith sequential component of $\overline{P}$ will make a (probabilistic) transition leading to the derivative $\overline{P'}$.*

The main result of this chapter compares the steady-state solution of a PEPA process with the steady-state solution of a related gPEPA process. The processes are equivalent where possible, except of course, the semantics of PEPA does not construct any probabilistic relations. Assume the ith sequential component $\overline{S_i}$ of some derivative $\overline{P}$ enables a set of generally distributed activities. The probabilistic semantics of gPEPA model this as an initial choice over probabilistic transitions, leading to a linear multi-stage process working off exponentially distributed lifetimes, leading to a final probabilistic choice of successor derivative. The main result requires that the mean lifetime of component $\overline{S_i}$ is equal to the mean lifetime of S_i.

Assume the stochastic process underlying the gPEPA model $\overline{P} \equiv (\overline{S_1}, \ldots, \overline{S_n})$ has a steady-state probability distribution, denoted by $\pi(.)$. Then the following global balance equation must hold:

$$
\begin{aligned}
\pi(\overline{P}) \Bigg(& \overbrace{\sum_{\substack{i=1 \\ S_i \text{ is exp}}}^{n} \sum_{\overline{P'} \in ds_i^{\longrightarrow}(\overline{P})} q(\overline{P}, \overline{P'}, i) \cdot \sum_{\overline{P''} \in ds_i^{\dashrightarrow}(\overline{P'})} Pr(\overline{P'}, \overline{P''})}^{\lambda \rightarrow new\ r} + \overbrace{\sum_{\substack{i=1 \\ \overline{S_i} \text{ has } k>1 \text{ lives}}}^{n} \nu_{\overline{S_i}}}^{r \rightarrow r-1} \\
& + \overbrace{\sum_{\substack{i=1 \\ \overline{S_i} \text{ has 1 life}}}^{n} \sum_{\substack{\overline{P'} \in ds_i^{\longrightarrow}(\overline{P}) \\ \overline{S_i'} \text{ is gen}}} \nu_{\overline{S_i}} \cdot Pr(\overline{P}, \overline{P'}, i)}^{r=1 \rightarrow new\ r} + \overbrace{\sum_{\substack{i=1 \\ \overline{S_i} \text{ has 1 life}}}^{n} \sum_{\substack{\overline{P'} \in ds_i^{\longrightarrow}(\overline{P}) \\ \overline{S_i'} \text{ is exp}}} \nu_{\overline{S_i}} \cdot Pr(\overline{P}, \overline{P'}, i)}^{r=1 \rightarrow \lambda} \\
& + \overbrace{\sum_{\substack{R \in 2^S \\ |R|>1}} \sum_{\substack{\overline{P'} \in ds_R^{\longrightarrow}(\overline{P}) \\ S_i \text{ is exp}, i \in R}} q(\overline{P}, \overline{P'}, R)}^{\lambda^n \rightarrow \lambda^n} + \overbrace{\sum_{\substack{R \in 2^S \\ |R|>1}} \sum_{\substack{\overline{P''} \in ds_R^{\longrightarrow}(\overline{P}) \\ S_i \text{ is gen} \\ S_j \text{ is exp},\ j \in R-i}} \sum_{\overline{P'} \in ps_i^{\longrightarrow}(\overline{P''})} \Pr(\overline{P''}, \overline{P'})}^{\lambda^n \rightarrow \lambda^{n-1}/new\ r} \Bigg) =
\end{aligned}
$$

$$
\begin{aligned}
&\overbrace{\sum_{i=1}^{n}\sum_{\overline{P''}\in ps_i^{\dashrightarrow}(\overline{P})}\sum_{\overline{P'}\in ds_i^{\dashrightarrow}(\overline{P''})}\pi(\overline{P'})\cdot q(\overline{P'},\overline{P''},i)\cdot \Pr(\overline{P''},\overline{P})}^{\lambda\to new\ r}+\overbrace{\sum_{i=1}^{n}\sum_{\substack{\overline{P'}\in ds_i^{\dashrightarrow}(\overline{P})\\ S_i' \text{ has } k>1 \text{ lives}}}\pi(\overline{P'})\cdot \nu_{S_i'}}^{r\to r-1}\\
&+\overbrace{\sum_{i=1}^{n}\sum_{\overline{P''}\in ps_i^{\dashrightarrow}(\overline{P})}\sum_{\substack{\overline{P'}\in ds_i^{\dashrightarrow}(\overline{P''})\\ S_i' \text{ has } 1 \text{ life}}}\pi(\overline{P'})\cdot \nu_{S_i'}\cdot \Pr(\overline{P''},\overline{P})}^{r=1\to new\ r}+\overbrace{\sum_{\substack{R\in 2^S\\ |R|>1}}\sum_{\overline{P'}\in ps_R^{\longrightarrow}(\overline{P})}\pi(\overline{P'})\cdot q(\overline{P'},\overline{P},R)}^{\lambda^n\to\lambda^n}\\
&+\overbrace{\sum_{i=1}^{n}\sum_{\substack{\overline{P'}\in ds_i^{\dashrightarrow}(\overline{P})\\ S_i' \text{ has } 1 \text{ life}}}\pi(\overline{P'})\cdot \nu_{S_i'}}^{r=1\to\lambda}+\overbrace{\sum_{i=1}^{n}\sum_{\substack{R\in 2^S\\ |R|>1, i\in R}}\sum_{\overline{P''}\in ps_i^{\dashrightarrow}(\overline{P})}\sum_{\overline{P'}\in ps_R^{\longrightarrow}(\overline{P''})}\pi(\overline{P'})\cdot q(\overline{P'},\overline{P''},R)\cdot \Pr(\overline{P''},\overline{P})}^{\lambda^n\to\lambda^{n-1}/new\ r}\\
&\qquad\text{where } S=\{1,\ldots,n\} \qquad (5.3.4)
\end{aligned}
$$

The label above each term is a mnemonic, intended to describe the terms contributing to the probability flux. Briefly, these are

- $\lambda \to$ *new* r: the flux due to a sequential component $\overline{S_i}$, enabling exponential activities, performing an individual activity leading the component $\overline{S_i''}$, which then completes a probabilistic transition to the linear component $\overline{S_i'}$. Component $\overline{S_i''}$ represents the translation of a component which enabled a generally distributed activity.

- $r \to r-1$: the flux due to a linear sequential component $\overline{S_i}$ completing one (exponential) stage.

- $r = 1 \to$ *new* r: the flux due to a linear sequential component $\overline{S_i}$ completing its last stage, and making a transition to $\overline{S_i''}$; this then completes a probabilistic transition to the linear component $\overline{S_i'}$. Component $\overline{S_i''}$ represents the translation of a component which enabled a generally distributed activity.

- $r = 1 \to \lambda$: the flux due to a linear sequential component $\overline{S_i}$ completing its last stage, and making a transition to $\overline{S_i'}$. Component $\overline{S_i'}$ enables exponentially distributed activities.

- $\lambda^n \to \lambda^n$: the flux due to a cooperation between sequential components leading to a new state in which the derivatives of all components which participated themselves enable exponential activities. Recall that cooperation is only possible on exponentially distributed activities.

- $\lambda^n \to \lambda^{n-1}/$*new* r: the flux due to a cooperation between sequential components, leading to a new state in which one sequential component newly enables a generally distributed activity.

5.3.1 The Proposition

Proposition 5.3.1 below shows that the gPEPA process $\overline{P}$, enabling generally distributed activities in a specifically constrained way, will have the same steady-state solution as P, as long as those sequential components which are generalised in $\overline{P}$ have sequential local balance in P.

Theorem 5.3.1. *Let $\overline{P}$ be a gPEPA process. For each sequential component $\overline{S_i}$ of $\overline{P}$, let $G_{\overline{P}}(\cdot)$ be defined as:*

$$G_{\overline{P}}(i) = \begin{cases} H_{\overline{S_i}}(k) & \textit{if } \overline{S_i} \textit{ has } k \textit{ Erlang mixture lifetimes left,} \\ 1 & \textit{otherwise.} \end{cases}$$

Assume that each generalised sequential component $\overline{S_i}$ has sequential local balance. Then it will be shown that:

$$\pi(\overline{S_1}, \ldots, \overline{S_n}) = \pi(\overline{P}) = \pi(P) \prod_{j=1}^{n} G_{\overline{P}}(j)$$

Proof

A case analysis is sufficient to show the required result. The global balance equation for the stochastic process underlying $\overline{P}$ is given by Equation (5.3.4). The proof proceeds by substituting $\pi(P) \prod_{j=1}^{n} G_{\overline{P}}(j)$ for $\pi(\overline{P})$, and individually considering two sets of terms for each component $\overline{P}$: those due to generalised sequential components, and those due to exponential sequential components.

The proof is split into two cases; the nature of the balance in the first case differs from that in the second.

Case 1: $\{\overline{S_i} : 1 \leq i \leq n\}$ When viewed as a probabilistic process, this case captures the balance between sequential components which are currently acting as linear PEPA processes, as the result of the translation of one or more generally distributed activities.

Equations (5.3.5) isolate the terms of the global balance equation which contribute.

$$
\begin{aligned}
LHS = \pi(P)\prod_{j=1}^{n} G_{\overline{P}}(j)\Bigg(&\overbrace{\sum_{\substack{i=1\\ \overline{S_i}\text{ has }k>1\text{ lives}}}^{n} \nu_{\overline{S_i}}}^{r\to r-1 \quad (1)} + \overbrace{\sum_{\substack{i=1\\ \overline{S_i}\text{ has 1 life}}}^{n}\ \sum_{\substack{\overline{P'}\in ds_i^{\longrightarrow}(\overline{P})\\ \overline{S_i'}\text{ is gen}}} \nu_{\overline{S_i}}\cdot Pr(\overline{P},\overline{P'},i)}^{r=1\to new\ r \quad (2)} \\
&+ \overbrace{\sum_{\substack{i=1\\ \overline{S_i}\text{ has 1 life}}}^{n}\ \sum_{\substack{\overline{P'}\in ds_i^{\longrightarrow}(\overline{P})\\ \overline{S_i'}\text{ is exp}}} \nu_{\overline{S_i}}\cdot Pr(\overline{P},\overline{P'},i)}^{r=1\to\lambda \quad (3)}\Bigg)
\end{aligned}
$$

$$
\begin{aligned}
RHS = &\overbrace{\sum_{i=1}^{n}\sum_{\substack{R\in 2^S\\ |R|>1}}\ \sum_{\overline{P}'\in ps_i^{\dashrightarrow}(\overline{P})}\ \sum_{\overline{P}''\in ps_{R\cup\{i\}}^{\longrightarrow}(\overline{P}')} \pi(\overline{P}'')\prod_{j=1}^{n} G_{\overline{P}''}(j)\cdot q(\overline{P''},\overline{P'},R\cup\{i\})\cdot Pr(\overline{P'},\overline{P},i)}^{\lambda^n\to\lambda^{n-1}/new\ r \quad (4)} \\
&+\overbrace{\sum_{i=1}^{n}\ \sum_{\overline{P''}\in ps_i^{\dashrightarrow}(\overline{P})}\ \sum_{\overline{P'}\in ds_i^{\dashrightarrow}(\overline{P''})} \pi(\overline{P'})\cdot q(\overline{P'},\overline{P''},i)\cdot \Pr(\overline{P''},\overline{P})}^{\lambda\to new\ r(5)} \\
&+\overbrace{\sum_{i=1}^{n}\ \sum_{\substack{\overline{P'}\in ps_i^{\longrightarrow}(\overline{P})\\ \overline{S_i'}\text{ has }k>1\text{ life}}} \pi(\overline{P}')\prod_{j=1}^{n} G_{\overline{P}'}(j)\cdot\nu_{\overline{S_i'}}}^{r\to r-1 \quad (6)} \\
&+\overbrace{\sum_{i=1}^{n}\ \sum_{\overline{P'}\in ps_i^{\dashrightarrow}(\overline{P})}\ \sum_{\substack{\overline{P''}\in ps_i^{\longrightarrow}(\overline{P'})\\ \overline{S_i''}\text{ has 1 life}}} \pi(\overline{P}'')\prod_{j=1}^{n} G_{\overline{P}''}(j)\cdot\nu_{\overline{S_i''}}\cdot Pr(\overline{P'},\overline{P},i)}^{r=1\to new\ r \quad (7)}
\end{aligned}
$$

where $S = \{1,\dots,n\}$ (5.3.5)

It is possible to immediately simplify some of these terms. First, consider terms (2) and (3). By Lemma 5.2.2,

$$\sum_{\overline{P'}\in ds_i^{\longrightarrow}(\overline{P})} Pr(\overline{P},\overline{P'},i) = 1 \tag{5.3.6}$$

This should intuitively be the case, and is deliberately so by construction; a condition of a correct probabilistic interpretation as Erlang mixtures is that the sum of all probabilistic options on leaving a state is equal to 1. It is therefore possible to simplify LHS to

$$
\begin{aligned}
LHS &= (1) + (2) + (3) \\
&= \pi(P) \prod_{j=1}^{n} G_{\overline{P}}(j) \Bigg(\sum_{\substack{i=1 \\ \overline{S_i} \text{ has } k>1 \text{ lives}}}^{n} \nu_{\overline{S_i}} + \sum_{\substack{i=1 \\ \overline{S_i} \text{ has 1 life} \\ d(\overline{S_i}) \text{ is gen}}}^{n} \nu_{\overline{S_i}} + \sum_{\substack{i=1 \\ \overline{S_i} \text{ has 1 life} \\ d(\overline{S_i}) \text{ is exp}}}^{n} \nu_{\overline{S_i}} \Bigg) \\
&= \pi(P) \prod_{j=1}^{n} G_{\overline{P}}(j) \sum_{\substack{i=1 \\ S_i \text{ has lives}}}^{n} \nu_{\overline{S_i}}
\end{aligned} \tag{5.3.7}
$$

Next, consider term (6). The component $\overline{P'}$ is in $ps_i^{\longrightarrow}(\overline{P})$ and furthermore, $\overline{S_i'}$ has more than one remaining lifetime. Therefore $\overline{P} = \overline{P'}$ and the following simplification is possible.

$$
\pi(\overline{P}') \prod_{j=1}^{n} G_{\overline{P}'}(j) = \pi(P) \prod_{\substack{j=1 \\ j \neq i}}^{n} G_{\overline{P}}(j) \cdot G_{\overline{P'}}(i) \tag{5.3.8}
$$

Further, notice that $\nu_{\overline{S_i}} = \nu_{\overline{S_i'}}$ since they are both part of the same sequential process (H, k). This leads to the following amalgamation.

$$
\begin{aligned}
(6) - (1) - (2) - (3) &= \sum_{i=1}^{n} \sum_{\substack{\overline{P'} \in ps_i^{\longrightarrow}(\overline{P}) \\ \overline{S_i'} \text{ has } k>1 \text{ life}}} \pi(P) \prod_{\substack{j=1 \\ j \neq i}}^{n} G_{\overline{P}}(j) \cdot G_{\overline{P'}}(i) \cdot \nu_{\overline{S_i'}} \\
&\quad - \sum_{\substack{i=1 \\ S_i \text{ has lives}}}^{n} \pi(P) \prod_{j=1}^{n} G_{\overline{P}}(j) \cdot \nu_{\overline{S_i}} \\
&= \sum_{\substack{i=1 \\ S_i \text{ has lives}}}^{n} \pi(P) \prod_{\substack{j=1 \\ j \neq i}}^{n} G_{\overline{P}}(j) \cdot (G_{\overline{P'}}(i) - G_{\overline{P}}(i)) \cdot \nu_{\overline{S_i}}
\end{aligned} \tag{5.3.9}
$$

since $|ps_i^{\longrightarrow}(\overline{P})| = 1$ and $\overline{S_i'}$ is generalised with more than 1 lifetime. Now, the term $G_{\overline{P'}}(i) - G_{\overline{P}}(i)$ is equivalent to

$$
H_{\overline{S_i}}(k) - H_{\overline{S_i}}(k-1) = -\frac{1}{\nu_{\overline{S_i}} \tau_{\overline{S_i}}} \cdot Pr(S_i \text{ starts with } k \text{ lives}) \tag{5.3.10}
$$

where as described in Lemma 5.2.1, $\tau_{\overline{S_i}}$ is the mean time until the end of the sojourn due to generally distributed activities enabled by $\overline{S_i}$. Therefore by construction of the stochastic process, $1/\tau_{\overline{S_i}}$ is given by $\sum_{\overline{P'} \in ds_i^{\longrightarrow}(\overline{P})} q(P, P', i)$. This finally leads to

Equation (5.3.11).

$$
\begin{aligned}
&(6)-(1)-(2)-(3)\\
=&\sum_{\substack{i=1\\ S_i \text{ has lives}}}^{n} \pi(P)\prod_{\substack{j=1\\ j\neq i}}^{n} G_{\overline{P}}(j)\cdot\Big(-1/\nu_{\overline{S_i}}\cdot Pr(S_i \text{ starts with } k \text{ lives})\sum_{P'\in ds_i^{\longrightarrow}(P)} q(P,P',i)\Big)\nu_{\overline{S_i}}\\
&=-\sum_{\substack{i=1\\ S_i \text{ has lives}}}^{n}\prod_{\substack{j=1\\ j\neq i}}^{n} G_{\overline{P}}(j)\cdot Pr(S_i \text{ starts with } k \text{ lives})\Big(\pi(P)\sum_{P'\in ds_i^{\longrightarrow}(P)} q(P,P',i)\Big)
\end{aligned}
\tag{5.3.11}
$$

Next, terms (4), (7) and (5) can be combined. Term (7) simplifies via Equation (5.3.12).

$$
\begin{aligned}
(7) &= \sum_{i=1}^{n}\sum_{\overline{P'}\in ps_i^{\dashrightarrow}(\overline{P})}\sum_{\substack{\overline{P''}\in ps_i^{\longrightarrow}(\overline{P'})\\ \overline{S_i''} \text{ has 1 life}}} \pi(\overline{P''})\prod_{j=1}^{n} G_{\overline{P''}}(j)\cdot\nu_{\overline{S_i''}}\cdot Pr(\overline{P'},\overline{P})\\
&= \sum_{i=1}^{n}\sum_{\overline{P'}\in ps_i^{\dashrightarrow}(\overline{P})}\sum_{\substack{\overline{P''}\in ps_i^{\longrightarrow}(\overline{P'})\\ \overline{S_i''} \text{ has 1 life}}} \pi(P'')\prod_{\substack{j=1\\ j\neq i}}^{n} G_{\overline{P''}}(j)\cdot\big(1/\nu_{\overline{S_i''}}\big)q(\overline{P''},i)\cdot\nu_{\overline{S_i''}}\cdot Pr(\overline{P'},\overline{P},i)\\
&= \sum_{i=1}^{n}\sum_{\overline{P'}\in ps_i^{\dashrightarrow}(\overline{P})}\sum_{\substack{\overline{P''}\in ps_i^{\longrightarrow}(\overline{P'})\\ \overline{S_i''} \text{ has 1 life}}} \pi(P'')\prod_{\substack{j=1\\ j\neq i}}^{n} G_{\overline{P''}}(j)\cdot q(P',P,i)\cdot Pr(S_i \text{ starts with } k \text{ lives})
\end{aligned}
\tag{5.3.12}
$$

However, note that $\overline{P''}$ differs from $\overline{P}$ in the ith sequential component only; that $P''=P'$; and finally, that $|ps_i^{\dashrightarrow}(\overline{P})|=1$. This leads to the following.

$$
(7) = \sum_{i=1}^{n}\prod_{\substack{j=1\\ j\neq i}}^{n} G_{\overline{P}}(j)\cdot Pr(S_i \text{ starts with } k \text{ lives})\sum_{\substack{\overline{P'}\in ps_i^{\longrightarrow}(\overline{P})\\ \overline{S_i'} \text{ has 1 life}}} \pi(P')q(P',P,i)\cdot
\tag{5.3.13}
$$

Term (5) can be simplified using similar reasoning. Secondly, term (4) can be manipulated as illustrated in Equation (5.3.14).

$$
\begin{aligned}
(4) &= \sum_{i=1}^{n} \sum_{\substack{R\in 2^S \\ |R|>1}} \sum_{\overline{P}'\in ps_i^{\dashrightarrow}(\overline{P})} \sum_{\overline{P}''\in ps_{\overrightarrow{R\cup\{i\}}}(\overline{P}')} \boldsymbol{A} \\
&= \sum_{i=1}^{n} \sum_{\substack{R\in 2^S \\ |R|>1}} \sum_{\overline{P'}\in ps_i^{\dashrightarrow}(\overline{P})} \sum_{\overline{P''}\in ps_{\overrightarrow{R\cup\{i\}}}(\overline{P'})} \boldsymbol{B} \\
\text{where } \boldsymbol{A} &= \pi(\overline{P}'') \prod_{j=1}^{n} G_{\overline{P}''}(j) \cdot q(\overline{P''}, \overline{P'}, R\cup\{i\}) \cdot Pr(\overline{P'}, \overline{P}, i) \\
\boldsymbol{B} &= \pi(P'') \prod_{\substack{j=1 \\ j\neq i}}^{n} G_{\overline{P''}}(j) \cdot q(P'', P', R\cup\{i\}) \cdot Pr(S_i \text{ starts with } k \text{ lives}) \\
&\quad \text{(by similar reasoning to Equation (5.3.13))} \\
&= \sum_{i=1}^{n} \prod_{\substack{j=1 \\ j\neq i}}^{n} G_{\overline{P}}(j) \cdot Pr(S_i \text{ starts with } k \text{ lives}) \sum_{\substack{R\in 2^S \\ |R|>1}} \sum_{P'\in ps_{\overrightarrow{R\cup\{i\}}}(P)} \boldsymbol{C} \\
\text{where } \boldsymbol{C} &= \pi(P') \cdot q(P', P', R\cup\{i\})
\end{aligned}
\tag{5.3.14}
$$

Terms (4), (7) and (5) can be added directly to give the following.

$$
\begin{aligned}
(4) + (7) &= \sum_{i=1}^{n} \prod_{\substack{j=1 \\ j\neq i}}^{n} G_{\overline{P}}(j) \cdot Pr(S_i \text{ starts with } k \text{ lives}) \sum_{R\in 2^S} \sum_{P'\in ps_{\overrightarrow{R\cup\{i\}}}(P)} \boldsymbol{A} \\
\text{where } \boldsymbol{A} &= \pi(P') \cdot q(P', P', R\cup\{i\})
\end{aligned}
\tag{5.3.15}
$$

Finally for this first case, all the simplified terms are amalgamated to produce Equation (5.3.16).

$$
\begin{aligned}
&(5.3.11) + (5.3.15) \\
&= -\sum_{\substack{i=1 \\ S_i \text{ has lives}}}^{n} \prod_{\substack{j=1 \\ j\neq i}}^{n} G_{\overline{P}}(j) \cdot Pr(S_i \text{ starts with } k \text{ lives}) \Bigg(\pi(P) \sum_{P'\in ds_i^{\longrightarrow}(P)} q(P, P', i) \Bigg) \\
&+ \sum_{i=1}^{n} \prod_{\substack{j=1 \\ j\neq i}}^{n} G_{\overline{P}}(j) \cdot Pr(S_i \text{ starts with } k \text{ lives}) \sum_{R\in 2^S} \sum_{P'\in ps_{\overrightarrow{R\cup\{i\}}}(P)} \pi(P') \cdot q(P', P', R\cup\{i\}) \\
&= \sum_{i=1}^{n} \prod_{\substack{j=1 \\ j\neq i}}^{n} G_{\overline{P}}(j) \cdot Pr(S_i \text{ starts with } k \text{ lives}) \boldsymbol{B} \\
&\text{where } \boldsymbol{B} = -\pi(P) \sum_{P'\in ds_i^{\longrightarrow}(P)} q(P, P', i) + \sum_{R\in 2^S} \sum_{P'\in ps_{\overrightarrow{R\cup\{i\}}}(P)} \pi(P') q(P', P, R\cup\{i\})
\end{aligned}
\tag{5.3.16}
$$

Term $\boldsymbol{B}$ in Equation (5.3.16) matches the sequential local balance equation given in (5.3.3), and since this is true by assumption, it can be deduced that $LHS = RHS$ in this case.

Case 2: $\{S_i : 1 \leq i \leq n\}$ Equations (5.3.17) isolate the terms of the global balance equation which contribute by performing exponentially distributed activities only.

$$
\begin{aligned}
LHS = \pi(P)\prod_{j=1}^{n} G_{\overline{P}}(j)\Bigg(&\overbrace{\sum_{\substack{i=1\\ S_i \text{ is exp}}}^{n} \sum_{\overline{P'}\in ds_{\overrightarrow{i}}(\overline{P})} q(\overline{P},\overline{P'},i)\cdot \sum_{\overline{P''}\in ds_{\dashrightarrow{i}}(\overline{P'})} Pr(\overline{P'},\overline{P''})}^{\lambda\to new\ r\quad (1)} \\
&+\overbrace{\sum_{\substack{R\in 2^S\\ |R|>1}} \sum_{\substack{\overline{P'}\in ds_{\overrightarrow{R}}(\overline{P})\\ S_i \text{ is exp}, i\in R}} q(\overline{P},\overline{P'},R)}^{\lambda^n\to\lambda^n\quad (2)} + \overbrace{\sum_{\substack{R\in 2^S\\ |R|>1}} \sum_{\substack{\overline{P''}\in ds_{\overrightarrow{R}}(\overline{P})\\ S_i \text{ is gen}\\ S_j \text{ is exp},\ j\in R-i}} \sum_{\overline{P'}\in ps_{\overrightarrow{i}}(\overline{P''})} \Pr(\overline{P''},\overline{P'})}^{\lambda^n\to\lambda^{n-1}/new\ r\quad (3)}\Bigg) \\
RHS = &\overbrace{\sum_{i=1}^{n} \sum_{\substack{\overline{P'}\in ds_{\dashrightarrow{i}}(\overline{P})\\ \overline{S'_i} \text{ has 1 life}}} \pi(P')\prod_{j=1}^{n} G_{\overline{P'}}(j)\cdot \nu_{\overline{S'_i}}\cdot Pr(\overline{P'},\overline{P},i)}^{r=1\to\lambda\quad (4)} \\
&+\overbrace{\sum_{\substack{R\in 2^S\\ |R|>1}} \sum_{\overline{P'}\in ps_{\overrightarrow{R}}(\overline{P})} \pi(P')\prod_{j=1}^{n} G_{\overline{P'}}(j)\cdot q(\overline{P'},\overline{P},R)}^{\lambda^n\to\lambda^n\quad (5)}
\end{aligned}
\tag{5.3.17}
$$

First, consider LHS. Immediately terms (2) and (3) can be combined to produce the following.

$$
\begin{aligned}
(2)+(3) &= \sum_{\substack{R\in 2^S\\ |R|>1}} \sum_{\substack{\overline{P'}\in ds_{\overrightarrow{R}}(\overline{P})\\ S_i \text{ is exp}, i\in R}} q(\overline{P},\overline{P'},R) + \sum_{\substack{R\in 2^S\\ |R|>1}} \sum_{\substack{\overline{P''}\in ds_{\overrightarrow{R}}(\overline{P})\\ S_i \text{ is gen}\\ S_j \text{ is exp},\ j\in R-i}} \sum_{\overline{P'}\in ps_{\overrightarrow{i}}(\overline{P''})} \Pr(\overline{P''},\overline{P'}) \\
&= \sum_{\substack{R\in 2^S\\ |R|>1}} \sum_{\overline{P'}\in ds_{\overrightarrow{R}}(\overline{P})} q(\overline{P},\overline{P'},R)
\end{aligned}
\tag{5.3.18}
$$

Now LHS can be simplified as the following derivation illustrates.

$$
\begin{aligned}
LHS &= (1) + (5.3.18) \\
&= \pi(P) \prod_{j=1}^{n} G_{\overline{P}}(j) \Bigg(\sum_{\substack{i=1 \\ S_i \text{ is exp}}}^{n} \sum_{\overline{P'} \in ds_i^{\longrightarrow}(\overline{P})} q(\overline{P}, \overline{P'}, i) \cdot \sum_{\overline{P''} \in ds_i^{\dashrightarrow}(\overline{P'})} Pr(\overline{P'}, \overline{P''}) \\
&\quad + \sum_{\substack{R \in 2^S \\ |R|>1}} \sum_{\overline{P'} \in ds_R^{\longrightarrow}(\overline{P})} q(\overline{P}, \overline{P'}, R) \Bigg) \\
&= \pi(P) \prod_{j=1}^{n} G_{\overline{P}}(j) \Bigg(\sum_{\substack{i=1 \\ S_i \text{ is exp}}}^{n} \sum_{\overline{P'} \in ds_i^{\longrightarrow}(\overline{P})} q(\overline{P}, \overline{P'}, i) + \sum_{\substack{R \in 2^S \\ |R|>1}} \sum_{\overline{P'} \in ds_R^{\longrightarrow}(\overline{P})} q(\overline{P}, \overline{P'}, R) \Bigg) \\
&= \pi(P) \prod_{j=1}^{n} G_{\overline{P}}(j) \sum_{R \in 2^S} \sum_{\overline{P'} \in ds_R^{\longrightarrow}(\overline{P})} q(\overline{P}, \overline{P'}, R) \\
&= \pi(P) \prod_{j=1}^{n} G_{\overline{P}}(j) \sum_{R \in 2^S} \sum_{P' \in ds_R^{\longrightarrow}(P)} q(P, P', R)
\end{aligned}
\tag{5.3.19}
$$

Secondly, term (4) may be simplified as follows.

$$
\begin{aligned}
(4) &= \sum_{i=1}^{n} \sum_{\substack{\overline{P'} \in ds_i^{\dashrightarrow}(\overline{P}) \\ \overline{S'_i} \text{ has 1 life}}} \pi(P') \prod_{j=1}^{n} G_{\overline{P'}}(j) \cdot \nu_{\overline{S'_i}} Pr(\overline{P'}, \overline{P}, i) \\
&= \sum_{i=1}^{n} \sum_{\substack{\overline{P'} \in ds_i^{\dashrightarrow}(\overline{P}) \\ \overline{S'_i} \text{ has 1 life}}} \pi(P') \prod_{\substack{j=1 \\ j \neq i}}^{n} G_{\overline{P'}}(j) \cdot G_{\overline{P}}(i) \cdot \nu_{\overline{S'_i}} \big(q(P', P, i) / q(P', i) \big) \\
&= \sum_{i=1}^{n} \sum_{\substack{\overline{P'} \in ds_i^{\dashrightarrow}(\overline{P}) \\ \overline{S'_i} \text{ has 1 life}}} \pi(P') \prod_{\substack{j=1 \\ j \neq i}}^{n} G_{\overline{P'}}(j) \cdot q(P', P, i)
\end{aligned}
\tag{5.3.20}
$$

Furthermore, since all terms $\overline{P'} \in ds_i^{\dashrightarrow}(\overline{P})$ are such that $\overline{S'_i}$ has one remaining lifetime, the RHS can be further simplified to the following.

$$
(4) = \sum_{i=1}^{n} \prod_{\substack{j=1 \\ j \neq i}}^{n} G_{\overline{P}}(j) \sum_{P' \in ds_i^{\dashrightarrow}(P)} \pi(P') q(P', P, i) \tag{5.3.21}
$$

Then with a view to amalgamating the terms present in RHS, term (5) can be manipulated as follows.

$$\begin{aligned}
(5) &= \sum_{\substack{R\in 2^S\\|R|>1}} \sum_{\overline{P'}\in ps_{\overrightarrow{R}}(\overline{P})} \pi(P') \prod_{j=1}^{n} G_{\overline{P'}}(j)\cdot q(\overline{P'},\overline{P},R)\\
&\text{where } S=\{i : 1\le i\le n, S_i \text{ is exp}\}\\
&= \sum_{\substack{R\in 2^S\\|R|>1}} \sum_{\overline{P'}\in ps_{\overrightarrow{R}}(\overline{P})} \pi(P') \prod_{\substack{j=1\\j\notin S}}^{n} G_{\overline{P'}}(j)\cdot q(P',P,R)\\
&= \prod_{\substack{j=1\\j\notin S}}^{n} G_{\overline{P}}(j) \sum_{\substack{R\in 2^S\\|R|>1}} \sum_{\overline{P'}\in ps_{\overrightarrow{R}}(\overline{P})} \pi(P')\cdot q(P',P,R)\\
&= \prod_{\substack{j=1\\j\notin S}}^{n} G_{\overline{P}}(j) \sum_{\substack{R\in 2^S\\|R|>1}} \sum_{P'\in ps_{\overrightarrow{R}}(P)} \pi(P')\cdot q(P',P,R)
\end{aligned} \tag{5.3.22}$$

Now terms (4) and (5) can be added together.

$$\begin{aligned}
RHS &= (4)+(5)\\
&= \sum_{i=1}^{n} \prod_{\substack{j=1\\j\neq i}}^{n} G_{\overline{P}}(j) \sum_{P'\in ds_{\overset{\dashrightarrow}{i}}(P)} \pi(P')q(P',P,i)\\
&+ \prod_{\substack{j=1\\j\notin S}}^{n} G_{\overline{P}}(j) \sum_{\substack{R\in 2^S\\|R|>1}} \sum_{P'\in ps_{\overrightarrow{R}}(P)} \pi(P')\cdot q(P',P,R)\\
&= \prod_{\substack{j=1\\j\notin S}}^{n} G_{\overline{P}}(j) \sum_{i=1}^{n} \sum_{P'\in ds_{\overset{\dashrightarrow}{i}}(P)} \pi(P')q(P',P,i)\\
&+ \prod_{\substack{j=1\\j\notin S}}^{n} G_{\overline{P}}(j) \sum_{\substack{R\in 2^S\\|R|>1}} \sum_{P'\in ps_{\overrightarrow{R}}(P)} \pi(P')\cdot q(P',P,R)\\
&= \prod_{\substack{j=1\\j\notin S}}^{n} G_{\overline{P}}(j) \sum_{R\in 2^S} \sum_{P'\in ps_{\overrightarrow{R}}(P)} \pi(P')\cdot q(P',P,R)
\end{aligned} \tag{5.3.23}$$

It is assumed that sequential local balance holds for each sequential component $\overline{S_i}$ that is generalised. This implies that the following equation also holds (where S is defined to consider only generalised sequential components).

$$\prod_{j=1}^{n} G_{\overline{P}}(j)\pi(P) \sum_{P'\in ds_{\overrightarrow{i}}(P)} q(P,P',i) = \prod_{j=1}^{n} G_{\overline{P}}(j) \sum_{R\in 2^S} \sum_{P'\in ps_{\overrightarrow{R\cup\{i\}}}(P)} \pi(P')q(P',P,R\cup\{i\})$$
$$\text{where } S=\{j : 1\le j\le n, \overline{S_i} \text{ is gen}\}\backslash\{i\} \tag{5.3.24}$$

Since this equation holds for each $\overline{S_i}$, these terms are added to LHS and RHS respectively. Finally, LHS and RHS may be expressed as follows.

$$\begin{aligned} LHS &= \pi(P) \prod_{j=1}^{n} G_{\overline{P}}(j) \sum_{R \in 2^S} \sum_{P' \in ds_{\overrightarrow{R}}(P)} q(P, P', R) \\ RHS &= \prod_{j=1}^{n} G_{\overline{P}}(j) \sum_{R \in 2^S} \sum_{P' \in ps_{\overrightarrow{R}}(P)} \pi(P') \cdot q(P', P, R) \end{aligned} \tag{5.3.25}$$

However, global balance states that

$$\pi(P) \sum_{R \in 2^S} \sum_{P \in ds_{\overrightarrow{R}}(P)} q(P, P', R) = \sum_{R \in 2^S} \sum_{P' \in ps_{\overrightarrow{R}}(P)} \pi(P') q(P', P, R) \quad \text{where } S = \{1, \ldots, n\} \tag{5.3.26}$$

and therefore, the conclusion is that $LHS = RHS$, completing the theorem. □

5.4 Implications and Examples

Theorem 5.3.1 gives conditions under which generally distributed activities may be introduced into a PEPA process; in fact if these conditions hold, they guarantee that a gPEPA process will have an equivalent steady-state solution to its corresponding PEPA process if the means of the activity distributions are equal.

There are two clear applications for results of this kind. One was the main motivation for this work, which is to improve the expressiveness available to SPA modellers. The result gives conditions under which non-exponential model elements may be introduced without impacting the tractability of model solution. The second application is in model aggregation. The distribution of time to absorption of a Markov chain is given by a phase-type distribution. An area of the state space of the Markov process can be considered as an absorbing Markov chain, and therefore replaced by a single activity with an appropriately calculated mean. If the component enabling the activity has sequential local balance, then the solution of the reduced model is equivalent to that of the original, in the sense that the steady-state probability of being present in the state enabling the aggregate activity is equal to the sum of the probabilities of being in the original aggregate states.

The result is distinct from previous work by Hillston which followed similar lines. In [44], the motivation is to construct a performance-preserving equivalence relation enabling a modeller to generate sufficient measures from smaller aggregated models. This was based around the contraction of reducible sequences, broadly speaking sequences of silent τ activities. The result given here is more general in allowing more arbitrary areas of the state space to be reduced as required (with the given restrictions). Furthermore, it extends the preliminary result detailed in [43] which catered for one generally distributed active element only. In a sense the theorem provides a reworking of Matthes theorem in a different mathematical setting (see Theorem 4.2.1). The constructed stochastic process is more complex in allowing countably infinitely many states, albeit in a restricted fashion. These are to cater for the introduction of general distributions, which are modelled by Erlang mixtures. However, the result guarantees that in grouping together states which correspond to the same generally distributed elements, the solution to the process is identical to the solution of its exponential partner.

The restrictions that were placed on the gPEPA models were:

- a sequential component enabling a generally distributed activity must have sequential local balance.
- two generally distributed activities cannot be newly enabled by two different sequential components at the same time.
- two generally distributed activities enabled by two different sequential components cannot both complete at the same time.
- two generally distributed activities may not cooperate.

In fact the second restriction can be understood as special cases of the first. Consider the following problematic gPEPA process.

$$\begin{array}{lcl@{\qquad}lcl} \overline{Q} & \stackrel{\mathrm{def}}{=} & (\alpha, r).\overline{Q'} & \overline{Q'} & \stackrel{\mathrm{def}}{=} & (\beta, X).\overline{Q''} \\ \overline{R} & \stackrel{\mathrm{def}}{=} & (\alpha, \top).\overline{R'} & \overline{R'} & \stackrel{\mathrm{def}}{=} & (\gamma, Y).\overline{R''} \\ \overline{P} & \equiv & \overline{Q} \underset{\{\alpha\}}{\bowtie} \overline{R} & \overline{P'} & \equiv & \overline{Q'} \underset{\{\alpha\}}{\bowtie} \overline{R'} \end{array} \tag{5.4.1}$$

Both $\overline{Q}$ and $\overline{R}$ represent different sequential components within $\overline{P}$. This process has only one option initially, that being to perform an α activity; after having done so, both of the generally distributed activities (β and γ) become enabled at the same time. Clearly $\overline{R}$ must be forbidden according to the restrictions above. The reason becomes clear on considering the required sequential balance equations for the exponential analogues of these components. Recall that these require that the flux out of a sequential component due to the completion of some activity is balanced by the enabling flux into that component. Consider the equations for $\overline{Q'}$ in the context $c = [.] \underset{\{\alpha\}}{\bowtie} \overline{R'}$, together with the global balance equations for $\overline{P'}$ given below.

$$\begin{array}{rcl}
\textit{Sequential Balance:} & & \\
\text{flux out of } c[Q] & = & \pi(P').E[X]^{-1} \\
\text{flux into } c[Q] & = & \pi(P).r \\
\textit{Global Balance:} & & \\
\text{flux out of } c[Q] & = & \pi(P').(E[X]^{-1} + E[Y]^{-1}) \\
\text{flux into } c[Q] & = & \pi(P).r
\end{array}$$

Clearly these equations are not consistent with each other.

The third restriction is an artefact of the construction of approximations to general distributions using Erlang mixtures, as explained in [24]. This example is made more interesting by comparison to the one given by Henderson and Lucic in [38] for stochastic Petri nets. Consider the SPN model given in Figure 5.4. The authors wish to

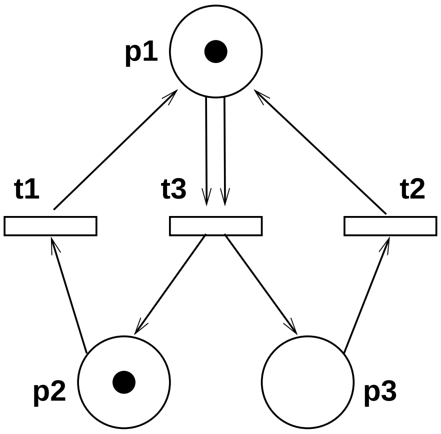

Figure 5.4: Insensitivity in a stochastic Petri net

consider insensitivity of SPN transitions; that is when a transition may have a generally distributed time-to-fire. They do this by constructing a GSMP model for the SPN, yielding a set of insensitivity balance equations for each transition. The equations for transition t_2 are:

$$\begin{array}{rcl}
\pi(0,1,1).E[t_2]^{-1} & = & \pi(2,0,0).E[t_1]^{-1} \\
\pi(1,1,0).E[t_2]^{-1} & = & 0
\end{array} \tag{5.4.2}$$

Regardless of the particular global balance equations, on the assumption that the model has a steady-state solution, the second equation cannot possibly be satisfied. Intuitively, the reason is that the marking $(1,1,0)$ represents a state which is such that in all of its predecessor markings, transition t_2 is already enabled. The same conclusion can be drawn from an intuitive and reasonable translation of Figure 5.4 into a gPEPA model. This translation is given below.

$$\begin{array}{rcl@{\qquad}rcl} \overline{L} & \stackrel{\text{def}}{=} & \mathtt{t}_1.\overline{L'} & \overline{L'} & \stackrel{\text{def}}{=} & \mathtt{t}_2.\overline{L} \\ \overline{R} & \stackrel{\text{def}}{=} & \mathtt{t}_1.\overline{R'} & \overline{R'} & \stackrel{\text{def}}{=} & \mathtt{t}_3.\overline{R} \\ \overline{\mathit{Trans}} & \equiv & \overline{L} \underset{\{\mathtt{t}_1\}}{\bowtie} \overline{R} & \overline{\mathit{Trans}'} & \equiv & \overline{L'} \underset{\{\mathtt{t}_1\}}{\bowtie} \overline{R} \end{array} \tag{5.4.3}$$

The SPN model behaves in such a way that transition t_2 is independent of transition t_3, and so it is reasonable to retain this structure in $\overline{\mathit{Trans}}$. Now let $c = [.] \underset{\{\mathtt{t}_1\}}{\bowtie} \overline{R}$; in similar fashion to earlier, the balance equations for this process can be obtained, with a view to determining the insensitivity of activity $\mathtt{t}_2$.

$$\boxed{\begin{array}{rcl} \multicolumn{3}{l}{\textit{Sequential Balance:}} \\ \text{flux out of } c[L'] & = & \pi(\mathit{Trans}').E[t_2]^{-1} \\ \text{flux into } c[L'] & = & \pi(\mathit{Trans}).E[t_1]^{-1} \\ \multicolumn{3}{l}{\textit{Global Balance:}} \\ \text{flux out of } c[L'] & = & \pi(\mathit{Trans}').(E[t_2]^{-1} + E[t_3]^{-1}) \\ \text{flux into } c[L'] & = & \pi(\mathit{Trans}).E[t_1]^{-1} \end{array}}$$

Via a sequential local balance analysis, the same conclusion is reached—that t_2 is not insensitive to its distribution.

5.4.1 Some Examples

A simple example of a legitimate gPEPA process is given below.

$$\overline{P_1} \stackrel{\text{def}}{=} (\alpha, r).(\beta, X).(\gamma, s).\overline{P_1} \tag{5.4.4}$$

$\overline{P_1}$ represents one sequential component only, and behaves as a simple cycle. It also meets each restriction listed at the start of this section. On consideration of the sequential local, and global, balance equations for its exponential counterpart, P_1, it is simple to see that they are consistent (indeed identical), and therefore $\overline{P_1}$ is insensitive to the distribution of β. This trivial example, and in fact any gPEPA model consisting of one sequential component, can be given a *semi-Markov process* representation; this stochastic model is insensitive to the residence time in any of its states. For example, $\overline{P_2}$ below also consists of one sequential component.

$$\overline{P_2} \stackrel{\text{def}}{=} (\alpha, X).(\mathtt{a}, r).\overline{P_2} + (\beta, Y).(\mathtt{b}, s).\overline{P_2} \tag{5.4.5}$$

Therefore once again, the sequential local balance equations are identical to those for global balance. However in this example, the balance equations do not obviously correspond to one activity—in fact there are two, α and β. What Theorem 5.3.1 says is that the process is insensitive to the residence time spent in $\overline{P_2}$, and by extension insensitive to the distributions associated with both α and β. However, if the aim is to solve this process assuming exponential distributions, then calculating the mean of $\min_{\{X,Y\}}$, and the time-averaged branching probabilities will be non-trivial in general.

In summary, this chapter presents balance equation conditions which guarantee the insensitivity of generally distributed activities within a gPEPA model. Restrictions are placed on the use of such activities; two may not be newly enabled simultaneously, and they may not be used to cooperate (synchronise). The continuous nature of the distributions ensures that two may not both complete at the same time. Under these restrictions, if a component has *sequential local balance*, then the process is insensitive to the distributions of its activities. Equivalently, it may be assumed that each generally distributed activity is exponentially distributed (with the same mean), and this stochastic process will provably have an identical steady-state solution. This would allow conventional Markovian solution techniques to be employed for a gPEPA process. Notice that the conditions for sequential local balance are not expressed as structural conditions on the form of the algebraic model.

Chapter 6

Case Study: Transaction Processing Systems

6.1 Introduction

In this chapter, both the reward structure methodology of Chapter 3 and the process algebra insensitivity structure of Chapter 5 are exemplified by the presentation of a case study. Although consistent with the literature, the results presented here may not be of novel interest, but rather are intended to highlight the use of the techniques presented earlier in the thesis.

The examples are drawn from the field of multi-user database systems (or *transaction processing systems* (TPS)). In recent years, the demand for these systems has grown rapidly, along with a strong focus on performance requirements. These systems consist of a centralised database, which controls a set of database *objects*; and a set of *transactions*, which attempt to access database objects, and thus perform, for example, user queries. Models of transaction processing systems abound in the literature; these include purely analytical approaches [50, 63], through to simulations [53].

Since the systems to be modelled are multi-user, and furthermore, in order to take advantage of technical developments like multiple processors, transactions will in general be viewed as behaving concurrently and independently, and thus may simultaneously attempt to interact with the same database objects. This raises the issue of the consistency of the database, in particular when more than one transaction wishes to update an object. The topic is known as *concurrency control* [72], and several methods exist to ensure the consistency of the database. The examples presented here, although idealised, incorporate concurrency control methods.

Transaction processing systems were chosen as the subject of this case study because although well-structured and suitable for an algebraic analysis, the calculation of performance measures is not a trivial task. It is shown that with some simplifying assumptions, it is possible to obtain a reasonable model of such a transaction processing system within the insensitive structure defined in Chapter 5. This means that the exponential assumption is unnecessary at various points in the model; or equivalently, allows a greater modelling freedom, whilst guaranteeing that with an exponential assumption always in place, the same solution will result.

This chapter begins by describing some relevant background to transaction processing systems, and highlights some important features with respect to the PEPA models. In Section 6.3, a simple model of a centralised database and a set of accessing transactions is presented. It is shown that the structure is insensitive, in the precise sense defined in Chapter 4, and thus the freedom to use generally distributed random variables gives a greater degree of realism. Section 6.4 proceeds with some alternative models, presented originally by Pun and Belford [63], and considers the problem of calculating useful performance measures. It is shown that by using the PEPA Reward language, it is possible to precisely specify, and automatically calculate, the measures required.

6.2 Transaction Processing Systems and Concurrency Control

The competition in industry such as finance and banking, coupled with the rapid advance of technology, has driven the modern development of large-scale transaction processing systems. In modern industrial applications, it is not unusual to find databases containing many tens of millions of records, or objects. Systems, such as those deployed by traders require rapid real-time access to data, in order to give them a vital edge. It is clear that the performance requirements on such database systems are stringent.

When transactions are able to update, or modify database objects, then an important consideration is maintaining the consistency of the data. The problem is well-known, and can be exhibited when the processing a transaction performs on a database object is not *atomic*. Consider a transaction which intends to update a particular datum; first, it reads the current value; a concurrent and independent transaction then accesses the same object, and *commits* an update (modification) to its value. The original transaction now calculates what it considers to be the correct new value, and *commits* its change. Clearly, the modification made by the first transaction has been lost. Preventing this problem from occurring, while ensuring the best possible performance of the database system, is the subject of concurrency control. A universally accepted correctness criterion for processing transactions against a database is *serializability*; that is that the interleaved execution of a set of concurrent transactions has an equivalent effect to their sequential execution [22]. One popular method for guaranteeing this property is known as *two-phase locking*. Several variations exist, and the performance of many have been studied in the literature. One variation incorporates *static locking* and *general waiting* [72]. Static locking means that a transaction may only be processed once locks have been obtained for all database objects that will be required (in contrast, dynamic locking means a transaction may request database objects as and when it needs them). However, in order to guarantee serializability, no locks are released until the transaction has completed its work in its entirety. General waiting means that a transaction that makes a lock request for a database object already exclusively locked by another transaction is blocked, awaiting the lock's release. Notice that if dynamic locking is used, this may result in a deadlock; two transactions may both be blocked, each awaiting the release of a lock that the other holds.

6.3 Modelling Transaction Processing Systems with PEPA

In this section, some PEPA definitions are presented to represent components of a transaction processing system. From these initial components, an insensitive model structure can immediately be realised; further development leads to more complex models with more involved behaviour. Nevertheless it is shown that the precise specification and extraction of performance measures can be handled by using the Reward language.

The TPS models consist of components to represent a collection of transaction classes, and a manager to enforce locking rules on database objects. Broadly, each transaction class may launch a transaction, which may then attempt to access particular database objects, be they pages or even records. In a typical database system, each object will be in a certain state of access—perhaps currently being modified by a transaction, or perhaps not being used at all. A model of a transaction may either attempt to

- modify a particular database object, in which case it attempts to acquire a *write* lock for that object; or
- read a particular object in its current state, in which case it does so regardless of any locks present on that object; this is called a *dirty read*.

Dirty reads are useful when a set of data is required often and in real-time. Although they run the risk of occasionally reading out-of-date information, they have the great advantage that they do not impact on transaction concurrency, and so keep transaction throughput higher. However, if a transaction acquires a write lock on an object, no other transaction may also acquire a write lock until the first has *committed* its changes.

This is modelled by forcing such transactions to block in an orderly fashion, sequentially gaining write access to the required object. According to the attributes of the class it belongs to, each transaction will choose to read from, or write to, an object with a fixed probability; and after the object access, another transaction of the same class will be immediately generated with another fixed probability. In this way, different transaction classes will be characterised in part by the mean number of transactions generated in a row.

Figure 6.1 presents a model which represents a class of transactions; that is, it can be used as part of a model when a sequence of transactions with particular characteristics are required. Component $TxnC_i$ represents the state in which no transactions of class

$$
\begin{array}{rcl}
TxnC_i & \stackrel{\text{def}}{=} & (\mathtt{think_i}, t_i).Txn_i \\
Txn_i & \stackrel{\text{def}}{=} & \sum_{j=1}^{N_i} (\mathtt{work}, w \times p_j).(\mathtt{writelock_{ij}}, r_{ij}).(\mathtt{commit_{ij}}, \top).Loop_i \\
 & + & (\mathtt{work}, r).(\mathtt{dirtyread}, d).(\mathtt{accessdb}, rl).Loop_i \\
Loop_i & \stackrel{\text{def}}{=} & (\mathtt{nexttxn_i}, r \times a_i).TxnC_i \\
 & + & (\mathtt{nexttxn_i}, (1 - r) \times a_i).Txn_i
\end{array}
$$

Figure 6.1: PEPA model of a transaction class

C_i are presently being processed. From this point, there is a delay, a *think* time (mean $1/t_i$), before the user generates a job which requires access to the database. The think time represents the delay between jobs generated by the supposed ith user or application. A transaction then begins the process of accessing the database by attempting either a write lock, or a dirty read, which is modelled by a probabilistic choice over the `work` activity. The issue of deadlock is avoided since in this model, transactions cannot accumulate locks—locks are released immediately the processing of an object is complete. Therefore this structure is closest to a *static* locking methodology [72].

Some assumptions are made on the respective database object requirements of each transaction class. Let D represent the set of all database objects; and further let $S_i = \{L_{i1}, \ldots, L_{iN_i}\}$ represent the class of objects to which write access is required by transactions characterised by class i. That is to say, a transaction of class i will require a set of locks L_{ij}, where $1 \leq j \leq N_i$. The choice over `work` probabilistically resolves the set of locks any particular class-i transaction will require. It is stipulated that each S_i partitions D in a non-overlapping fashion—that is, $\bigcap_{j=1}^{N_i} L_{ij} = \emptyset$. From this the assumption is made that if a class-i transaction requires a set of locks L_{ij}, that none of these locks are present in partition $L_{i'j'}$, for each transaction class $i \neq i'$, and each $j' \neq j$. In terms of the model, this means that a class-i transaction holding a particular set of locks cannot prevent the progress of a class-i' transaction wishing to take a set of locks from a different partition. Intuitively, this means that although in general, each transaction class will see a different partitioning of the set of objects, a class-i partition will only overlap with *one* class-i' partition. Although a strong assumption, this is not infeasible, as demonstrated in Figure 6.2. Assuming that the transaction

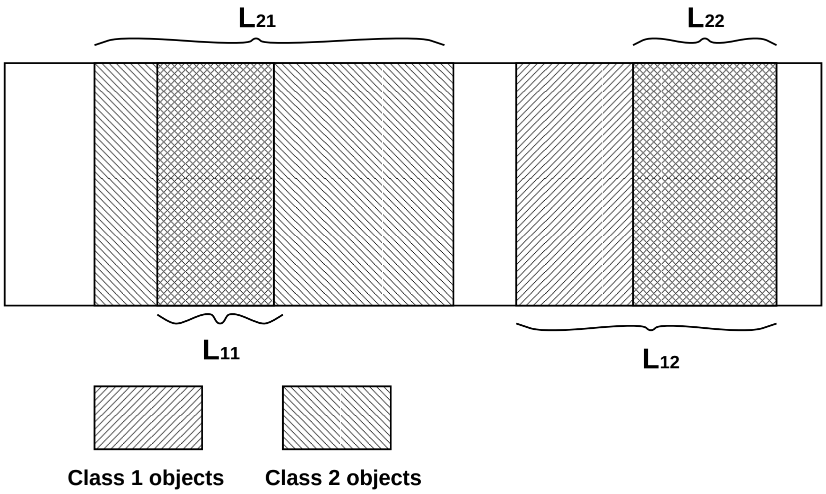

Figure 6.2: A valid transaction access pattern

requires exclusive locks (such that other transactions must wait to access the locked objects), an additional factor in the sum over `work` governs which area of the database the transaction wishes to access. In this way, the model captures the fact that parts of the database may be more frequently accessed than others. This phenomenon is known variously as the 'b-c' rule and '80-20' rule [72], meaning that b% (20%) of the database is accessed on average c% (80%) of the time. Activities `writelock` and `dirtyread` model the point at which the transaction attempts to access an object. If a write lock is awarded, the object is used, and the transaction commits its changes. The work of the current transaction is now done, and the model now probabilistically determines if the current job features more transactions. This is specified by a choice over $\mathtt{nexttxn_i}$, where r is the probability that the current job will generate more transactions. If required, another is generated, and if not, a transition is made back to $TxnC_i$, to await another job from the ith user.

The next requirement is to model the central database itself. This will be an abstraction, allowing access to database objects, and enforcing a two-phase locking concurrency control scheme. The database is constructed as a product of n processes which control access to M classes of write-locked database objects. Each of these lock manager processes, and the collection constituting the database, are given by expressions, the form of which appears in Figure 6.3. Each *LockMgr* process simply acts as a queue—

$$LockMgr_{j,\langle\rangle} \stackrel{\mathrm{def}}{=} \sum_{i=1}^{M} (\mathtt{writelock_{ij}}, \top).LockMgr_{j,\langle i\rangle}$$

$$LockMgr_{j,\langle k,a,\ldots,b\rangle} \stackrel{\mathrm{def}}{=} \sum_{\substack{i=1\\ i\notin\{k,a,\ldots,b\}}}^{M} (\mathtt{writelock_{ij}}, \top).LockMgr_{j,\langle k,a,\ldots,b,i\rangle}$$
$$+ (\mathtt{commit_{kj}}, c_j).LockMgr_{j,\langle a,\ldots,b\rangle} \qquad |\{k,a,\ldots,b\}| < n \tag{6.3.1}$$

Figure 6.3: The database lock managers

a transaction performing a `writelock` claims exclusive access to a particular object, and any other transactions making subsequent conflicting requests are blocked until, by performing a `commit`, the original claimant releases the required resource. This observation hints at the alternative way to model this system, to be described later. Figure 6.4 gives the definition of the transaction processing system in its entirety.

$$TPS \stackrel{\mathrm{def}}{=} (\prod_{i=1}^{M} TxnC_i) \underset{L}{\bowtie} (\prod_{j=1}^{n} LockMgr_{j,\langle\rangle})$$
$$\text{where } L = \{\mathtt{commit_{ij}}, \mathtt{accessdb_{ij}} : 1 \le i \le M, 1 \le j \le n\} \tag{6.3.2}$$

Figure 6.4: The complete transaction processing system

6.3.1 Exploiting an Insensitive Structure

There are elements of this model which it may not be appropriate to model using an exponential distribution. One such activity may be the think time of each user, i.e. $\mathtt{think_i}$, $1 \le i \le n$, for example if the 'user' in one case is a computer program generating transactions at a constant rate. In this case it is possible to choose a generally distributed duration of activity, since the TPS may be modelled by making use of the combinator presented in Chapter 4. Rate restriction is in place, since each lock manager controls the rate at which waiting transactions may commit changes to the database i.e. activities $\mathtt{commit_{kj}}$ happen at a constant rate. Therefore the rate is fixed independent of the number of blocked transactions, which is actually stricter than the queue-length dependent rates required. This is not unreasonable, since access times will depend on the database objects, not on the transactions themselves.

Proposition 1. *The steady-state solution of TPS with arbitrarily distributed* $\mathtt{think_i}, 1 \leq i \leq n$ *is identical to the steady-state solution of TPS where those activities are exponentially distributed (with the same mean of* $\frac{1}{t_i}$*).*

This proposition allows the transaction processing system to be modelled in PEPA, yet with no commitment to the stochastic nature of the distribution associated with each $\mathtt{think_i}$ activity other than the mean. The QD model, called TPS', consists of the same set of transaction class processes, but where the set of *LockMgr* processes is replaced by a queueing discipline. The formal definition of TPS' is given in Equation (6.3.1).

$$\begin{aligned} TPS' &\stackrel{\text{def}}{=} Q_\chi(TxnC_1, \ldots, TxnC_M) \\ &\text{where } \chi = \langle A_1, \xi_1, \ldots, A_N, \xi_N \rangle \\ \mathcal{A}_i &= \{\mathtt{writelock_{ij}} : 1 \leq i \leq M\}, 1 \leq j \leq n \\ \xi_i &= \langle c_i, \ldots, c_i \rangle \end{aligned} \tag{6.3.3}$$

These models are in fact isomorphic, that is $TPS = TPS'$. Given $Q_\chi(TxnC'_1, \ldots, TxnC'_M) \in ds(TPS')$, the steady-state solution of being present in $Q_\chi(TxnC'_1, \ldots, TxnC'_M)$ is given by the following expression:

$$\pi(Q_\chi(TxnC'_1, \ldots, TxnC'_M)) = \frac{1}{G}\prod_{i=1}^{M}\pi_i(TxnC'_i).\prod_{i=1}^{n}\prod_{j=q_i+1}^{d_j} c_j. \prod_{\substack{i=1 \\ TxnC'_i \text{ is blocked}}}^{M} \sum_{(\alpha,r)\in\mathcal{A}ct(TxnC'_i)} r \tag{6.3.4}$$

where $1/G$ is a normalising constant, and q_i represents the number of transactions blocked wishing to access database objects in L_{ki} for $1 \leq k \leq M$.

6.4 Using the PEPA Reward language

In this section, the PEPA Reward language is employed to specify and automatically calculate performance measures from a set of TPS models due to Pun and Belford [63]. These models are treated analytically by the authors, using queueing theory and the BCMP theorem. However, they serve to prove the main point of this section—that the Reward language can be used to automatically generate performance measures from these models. The results presented match those produced by Pun and Belford using an alternative analysis; the Reward language is then used to specify and calculate some more elaborate properties.

6.4.1 Description of the Models

Pun and Belford [63] present three classes of models for a TPS. Once again, transactions in this model obey two-phase locking, and employ a preclaim strategy for securing database objects. A database consists of a collection of data objects, bundled into *granules*. If a transaction acquires a lock over a data item, all data items in the granule are also locked. In these models, it is assumed that granules are 'well-placed', that is that the data items referenced by a transaction are packed into as few granules as possible. Following Pun and Belford, let TS be the number of items referenced by a transaction, DS the size of the database in items, and NG the number of granules in the database, then the number of items in each granule is $G = \lceil DS/NG \rceil$.

When a transaction enters the system, it requests all locks at once. The lock manager determines whether all these locks may be granted or not. If any of the locks are held by another transaction, the request cannot be granted, and the transaction is blocked. Otherwise, the locks are granted, and the transaction is allowed to access all its held data items. It then alternates between computation and database access, where each access retrieves one data item. After its final computation, the transaction terminates and all its locked data items are released. If another blocked transaction may now be unblocked, this is done, and it proceeds as described.

The model may be represented as a queueing network, as illustrated in Figure 6.5. The integer labels represent classes of customer—classes 1, 2, 3 and 4 pass through the

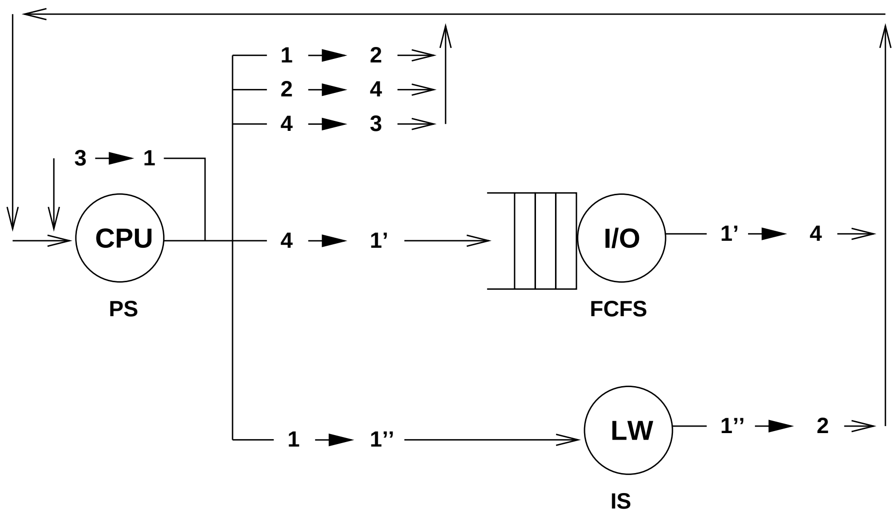

Figure 6.5: Queueing model representation of TPS

processor-sharing CPU node, and therefore represent some computation with respect to a transaction; class 1' represents a transaction accessing the database, customers being governed by a FCFS node; and class 1" represents a transaction waiting for locks, modelled by an IS (fixed delay) node. A class 1 customer is a transaction requesting locks, class 2 represents locks being granted and set, class 3 the locks being released, and class 4 represents computation by the transaction between database accesses.

Each transaction enters as a class 1 customer, and requests locks, which is either granted, causing a change to class 2, or denied causing a change to 1". A class 2 transaction is changed to class 4 on having its locks granted, and then cycles between the CPU and accessing the database, alternating between class 4 and class 1'. On the departure of a transaction, a new one immediately enters—Pun and Belford's model is *closed*.

Since the authors solve such models analytically, the movement between customer classes, and the delays incurred by the service centres are modelled probabilistically. Here, a concise summary is given—the interested reader is referred to [63] for more details. The models are parameterised by the following additional variables:

- N, the degree of multiprogramming, that is, the number of transactions concurrently executing.
- $TCPU$, the mean time spent by a transaction on a 'useful' computation.
- $TLKREQ$, the mean time spent deciding on a lock request.
- $TLKSET$, the mean time spent setting a lock.
- $TLKNREL$, the mean time spent releasing a lock (on transaction completion).
- TIO, the mean time taken by a transaction to access the database.
- $MTULG$, the mean wait time for a blocked transaction to obtain its locks.

Now, let $\mu_{i,r}$ denote the service rate of class r customers at service centre i. Let $P_{i,r;j,s}$ denote the probability that a class r customer at centre i will become a class s customer and next require service at centre j. Then the following parameters are chosen, which depend on a value P_g, the probability that a transactions locks are granted:

$$\begin{aligned}
1/\mu_{CPU,1} &= TLKREQ & 1/\mu_{CPU,2} &= TLKSET \\
1/\mu_{CPU,3} &= TLKNREL & 1/\mu_{CPU,4} &= TCPU/(TS+1) \\
1/\mu_{I/O,1'} &= TIO/TS & 1/\mu_{LW,1''} &= MTULG \\
P_{CPU,1;CPU,2} &= P_g & P_{CPU,1;LW,1''} &= 1 - P_g \\
P_{CPU,4;CPU,3} &= 1/(TS+1) & P_{CPU,4;I/O,1'} &= TS/(TS+1)
\end{aligned} \tag{6.4.1}$$

$$P_{CPU,1;CPU,4} = P_{CPU,3;CPU,1} = P_{I/O,1';CPU,4} = P_{LW,1'';CPU,2} = 1$$

and all other probabilities are assumed to be 0.

6.4.2 Representing the Model in PEPA

Modelling a transaction such that it is faithful to the behaviour of the queueing model above is reasonably straightforward. The definition is given below.

$$\begin{array}{rcl} \mathit{Txn}_{i1} & \stackrel{\mathrm{def}}{=} & (\mathtt{reqlocks}, \mu_{CPU,1} \times P_{CPU,1;CPU,2}).\mathit{Txn}_{i2} \\ & + & (\mathtt{reqlocks}, \mu_{CPU,1} \times P_{CPU,1;LW,1''}).(\mathtt{waitforlocks}, \mu_{LW,1''}).\mathit{Txn}_{i2} \\ \mathit{Txn}_{i2} & \stackrel{\mathrm{def}}{=} & (\mathtt{setlocks}, \mu_{CPU,2}).\mathit{Txn}_{i4} \\ \mathit{Txn}_{i4} & \stackrel{\mathrm{def}}{=} & (\mathtt{compute}_{\mathtt{i,DB}}, \mu_{CPU,4} \times P_{CPU,4;I/O,1'}).\mathit{Txn}_{i1'} \\ & + & (\mathtt{compute}_{\mathtt{i,rel}}, \mu_{CPU,4} \times P_{CPU,4;CPU,3}).\mathit{Txn}_{i3} \\ \mathit{Txn}_{i1'} & \stackrel{\mathrm{def}}{=} & (\mathtt{accessdb}_{\mathtt{i}}, \top).\mathit{Txn}_{i4} \\ \mathit{Txn}_{i3} & \stackrel{\mathrm{def}}{=} & (\mathtt{releaselocks}, \mu_{CPU,3}).\mathit{Txn}_{i1} \end{array} \tag{6.4.2}$$

Progress through a non FCFS service centre is modelled by a PEPA activity with an appropriately chosen rate; the probability of exchanging customer classes after a service centre is modelled by a choice (a *race*) between two activities, where the probabilities are introduced into the activity rates. The FCFS service centre is modelled in straightforward fashion as a PEPA queue:

$$\begin{array}{rcll} Q_{\langle\rangle} & \stackrel{\mathrm{def}}{=} & \sum_{i=1}^{N} (\mathtt{compute}_{\mathtt{i,DB}}, \top).Q_{\langle i\rangle} & \\ Q_{\langle k,a,\ldots,b\rangle} & \stackrel{\mathrm{def}}{=} & \sum_{i \notin \{k,a,\ldots,b\}} (\mathtt{compute}_{\mathtt{i,DB}}, \top).Q_{\langle k,a,\ldots,b,i\rangle} & \\ & + & (\mathtt{accessdb}_{\mathtt{k}}, \mu_{IO,1'}).Q_{\langle a,\ldots,b\rangle} & \text{where } |\{k,a,\ldots,b\}| < N \\ Q_{\langle k,a,\ldots,b\rangle} & \stackrel{\mathrm{def}}{=} & (\mathtt{accessdb}_{\mathtt{k}}, \mu_{IO,1'}).Q_{\langle a,\ldots,b\rangle} & \text{where } |\{k,a,\ldots,b\}| = N \end{array} \tag{6.4.3}$$

A benefit of this model is the insight it affords into parameterising the model by N, the degree of multiprogramming—the individual transactions are explicitly represented in the complete specification below:

$$\begin{array}{c} \mathit{TPS}_{PB} \stackrel{\mathrm{def}}{=} (\overbrace{\mathit{Txn}_{11} \parallel \ldots \parallel \mathit{Txn}_{N1}}^{N \text{ times}}) \underset{L}{\bowtie} Q_{\langle\rangle} \\ \text{where } L = \{\mathtt{compute}_{\mathtt{i,DB}}, \mathtt{accessdb}_{\mathtt{i}} : 1 \le i \le N\} \end{array} \tag{6.4.4}$$

Therefore the system is composed of the independent product of N transaction processes, in cooperation with a FCFS service centre.

6.4.3 Choosing Parameters

The aim of this study was to use the PEPA Reward language to reproduce the results reported in [63], and further to specify and calculate some more elaborate performance measures. Pun and Belford calculate the percentage of useful CPU utilisation, for different degrees of multiprogramming ($N = 2, 4, 6$), as NG, the number of granules in the database, and TS, the number of items required per transaction, are varied.

Before solving the models, concrete values were given to the model parameters. Let $NL = \lceil DS/G \rceil$ giving the number of locks required by a transaction. The parameters chosen were

$$
\begin{aligned}
DS &= 5000 \text{ items} & TCPU &= 50 \times TS \\
TLKREQ &= 5 \times NL & TLKSET &= 5 \times NL \\
TLKNREL &= 10 \times NL & TIO &= 50 \times TS
\end{aligned} \tag{6.4.5}
$$

Two parameters remain unspecified. The first is P_g, the probability that a transactions locks are granted. Following Pun and Belford, this is estimated in the following way. Let $A(k)$ be the probability that a new transaction is activated, given that k transactions are already in the system. To allow the transaction to be activated, the granules holding the data items it requests must be outside the $k \times NL$ granules already held by the k active transactions. The number of ways to choose these granules is $f(k) = \mathsf{choose}(NG - k \times NL, NL)$; the total number of ways to choose NL granules is $p = \mathsf{choose}(NG, NL)$; this gives $A(k) = f(k)/p$. To determine the average number of transactions in the system, NA, Pun and Belford use a well-founded recurrence relation determining the probability k transactions are active given m are in the system i.e.

$$NA = \sum_{k=1}^{N-1} k \times P(k \mid N-1)$$

ultimately determining $P_g = A(NA)$.

Each parameter so far has been calculated analytically, given an initial set of model conditions such as the number of data items, granule size, etc. The final value to be specified is $MTULG$, the mean time a transaction has to wait for locks to be granted. Let T be the throughput of transactions. When a transaction finishes, a blocked transaction is immediately activated, meaning the number of active transactions in the system remains the same. After a transaction joins the 'blocked' queue, the expected queue length is $N - NA$, giving an estimate for $MTULG$ of $(N - NA) \times 1/T$. However, in order to determine T, the model must be solved. This leads to an interative process.

The initial value chosen for $MTULG$ is the total resource time requested by each transaction, $TLKREQ + TLKSET + TLKNREL + TCPU + TIO$. Thereafter, the PEPA Reward language can be employed to automatically generate the transaction throughput T, allowing the value of $MTULG$ to be refined, and the process repeated until convergence. A suitable reward specification and attachment are given below:

$$\begin{aligned} \text{spec} &= (\Delta_{\texttt{setlocks}}, rate(\texttt{setlocks})) \\ \text{attachment} &= (\text{spec}, c, \langle \overbrace{Txn_{11}, \ldots, Txn_{N1}}^{N \text{ times}}, Q_{\langle\rangle} \rangle) \\ \text{where } c &= (\overbrace{[.] \mid\mid \ldots \mid\mid [.]}^{N \text{ times}}) \underset{L}{\bowtie} [.] \end{aligned} \tag{6.4.6}$$

This reward specification is satisfied by states in which an activity of type `setlocks` is enabled. By assigning to these states as a reward the rate at which `setlocks` is performed, and 0 to all others, the reward structure built gives the overall transaction throughput by the Forced Flow law. The PEPA Workbench was used to calculate the throughput, and the value was used iteratively in the construction and solution of subsequent refined models. As mentioned, the Reward language is only partially implemented—however, the reward above does not require an analysis of the behaviour of individual subcomponents, and so the Workbench could be satisfactorily used. Figure 6.4.3 shows the value of T converging for $TS = 250$, $N = 4$, and $NG = 8000$.

Iteration	Start Value	Resulting Value
1	17543	22222
2	22222	25641
3	25641	27777
4	27777	29411
5	29411	31250
6	31250	31250

Figure 6.6: Convergence of mean time between transactions

6.4.4 Specifying the Performance Measures

Once a final value of T had been derived, the next task was to calculate the useful CPU utilisation. This is defined as the time the CPU is being used by any transaction in between accesses to the database. By studying TPS_{PB}, it can be seen that any state from which an activity of type $\texttt{compute}_{\texttt{i,DB}}$ is possible, for any i such that $1 \leq i \leq N$, represents the utilisation of the CPU by a transaction. Therefore, the PEPA Workbench was provided with the following reward specification:

$$\text{spec} = (\Delta_{\texttt{compute}_{\texttt{1,DB}}} \vee \ldots \vee \Delta_{\texttt{compute}_{\texttt{N,DB}}}, 1) \tag{6.4.7}$$

Results were calculated for values of $N = 2, 3, 4$, $TS = 250, 5$, and with NG varying between 1 and 10000. The CPU utilisation is presented in Figures 6.7 and 6.8. These

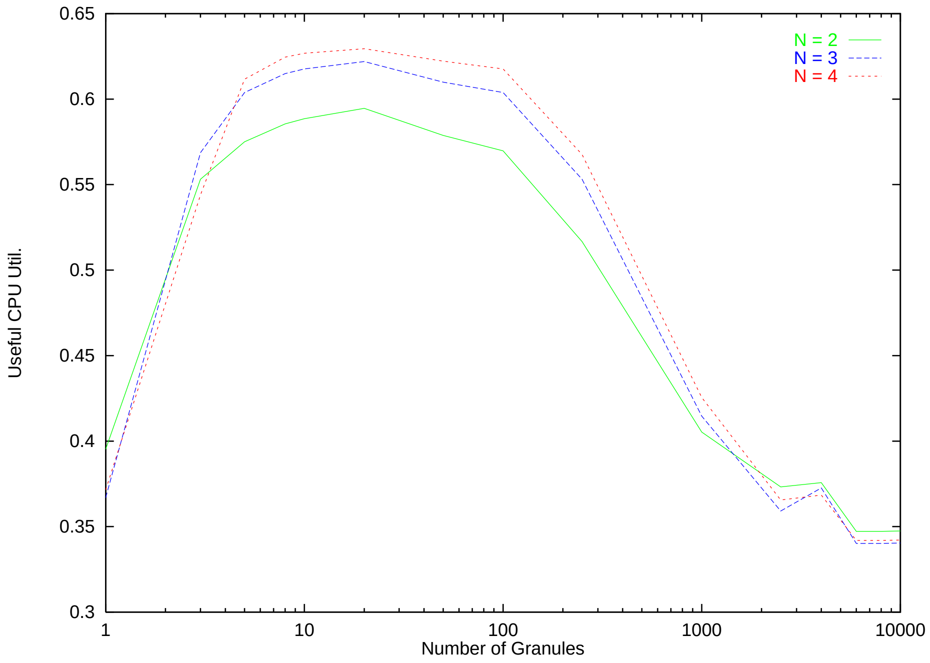

Figure 6.7: CPU utilisation for a transaction size of 250

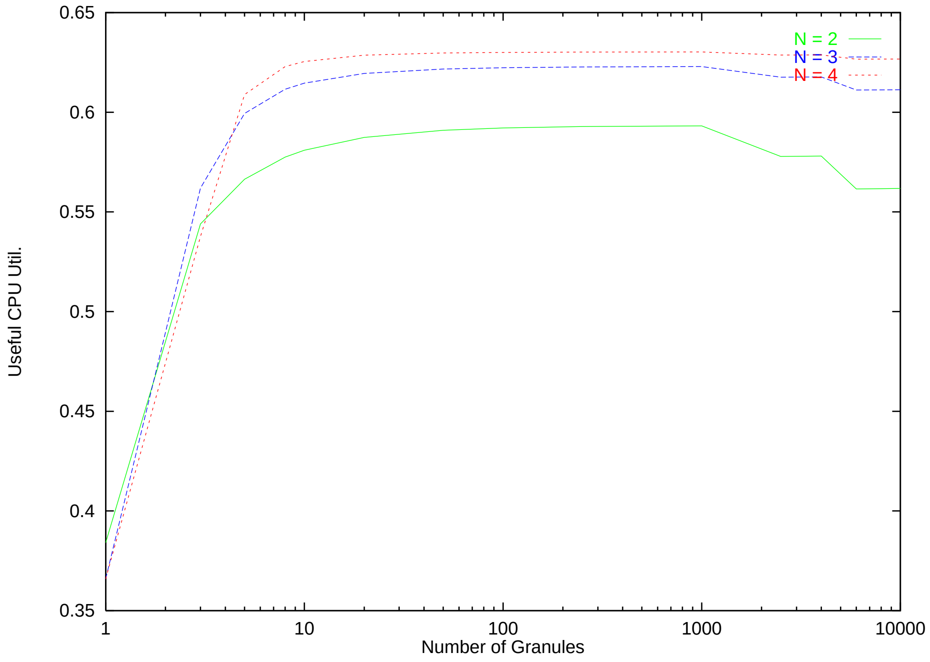

Figure 6.8: CPU utilisation for a transaction size of 5

results mirror those presented in [63]. Figure 6.7 shows that for a transaction size of 250, the utilisation hits a peak at around 20 granules. The curve rises as the number of granules in the database increases, which is intuitive since this will increase the number of active transitions in the system. However beyond $NG = 20$ it is the case that the number of locks required by each transaction, $\lceil DS/NG \rceil$, rises above 1, and thus the locking overhead increases. Figure 6.8 indicates that for a much lower transaction size, the CPU is effectively utilised at much higher values of NG, that is with much finer database granularity. However it also shows that beyond low values of NG, the increase in utilisation is not marked, and therefore the benefits of such fine granularity are slight.

The next measure specified is used in the calculation of the mean number of transactions waiting to access the database (therefore the mean number queueing at the FCFS service centre). In order to do this, each state of the model which represents i queueing transactions must be assigned a reward of i. This can be done by using several reward specifications; the case for $N = 2$ is shown below:

$$\begin{aligned}
\mathrm{spec}_0 &= (\nabla_{\texttt{accessdb}_1} \wedge \nabla_{\texttt{accessdb}_2}, 0) \\
\mathrm{spec}_1 &= ((\Delta_{\texttt{accessdb}_1} \wedge [\texttt{accessdb}_1]\nabla_{\texttt{accessdb}_2}) \\
&\vee \ (\Delta_{\texttt{accessdb}_2} \wedge [\texttt{accessdb}_2]\nabla_{\texttt{accessdb}_1}), 1) \\
\mathrm{spec}_2 &= (\langle\texttt{accessdb}_1\rangle\Delta_{\texttt{accessdb}_2} \vee \langle\texttt{accessdb}_2\rangle\Delta_{\texttt{accessdb}_1}, 2)
\end{aligned} \tag{6.4.8}$$

The expected number of queueing transactions can be calculated by simply adding together each of the reward structures produced from the specifications above. The results are presented in Figures 6.9 and 6.10. It can be seen that these graphs follow

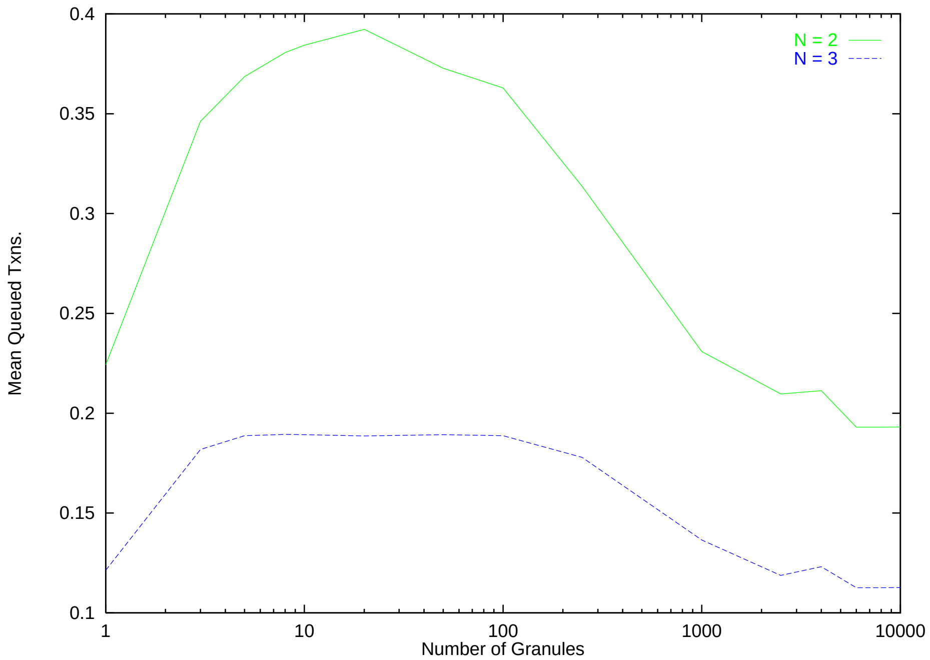

Figure 6.9: Expected number of queueing transactions for $TS = 250$

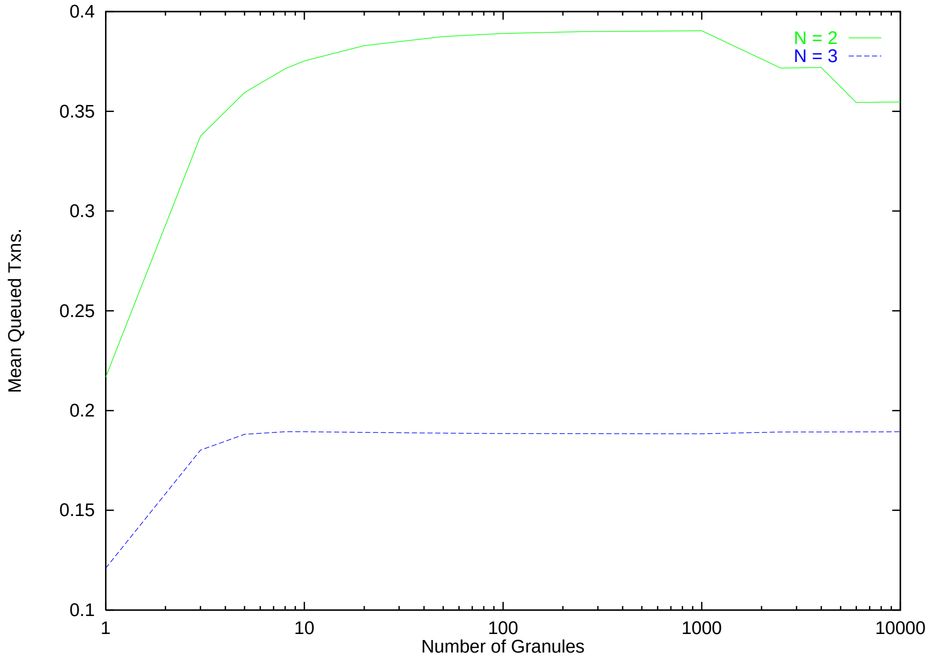

Figure 6.10: Expected number of queueing transactions for $TS = 5$

approximately similar trends to the previous two. This is for the same reason that while the granularity increases, this increases the locking overhead, and so more transactions are blocked, preventing them from gaining their locks in the first place. The curves rise initially because for low values of NG, each transaction only requires one lock per transaction, and so the locking overhead remains constant while allowing more transactions into the system at any one time.

6.5 Summary

This chapter has exhibited the two themes of this thesis. Firstly, a transaction processing system was shown to be suitable for modelling with the new combinator presented in Chapter 4. This meant that the model was insensitive to a particular set of activities. Some features of the TPS were unsuitable for modelling with exponential random variables, and it was shown that in these cases, the exponential assumption in the model was unnecessary—the corresponding activities could be distributed arbitrarily without affecting the steady-state solution of the model. Secondly, TPS models of Pun and Belford [63] were translated into PEPA, and it was shown that by use of the PEPA Reward language, reward structures that were specified logically, and calculated automatically, could produce the same performance measures as those calculated analytically from another modelling paradigm, networks of queues. Furthermore, the Reward language was used to calculate some more elaborate properties of the models.

Chapter 7

Conclusions

Extending and improving the repertoire of techniques available to the SPA modeller is a worthwhile pursuit. As well as assisting the performance modeller already convinced by, and working with, process algebra, it should also serve to persuade others that the methodology is maturing and may serve their needs. The author is in no doubt that the compositionality present in SPA gives a significant advantage over other modelling paradigms. Much of the work presented in this thesis exploits this compositionality, in both constructing and analysing stochastic process algebra models.

7.1 Summary

The theme of this thesis has been to build on model construction and analysis techniques for stochastic process algebras. This has been done in two ways—firstly, by providing a specification language for performance measures, and secondly, by providing conditions for incorporating generally distributed random variables into SPA models.

The PEPA Reward language was presented, a specification language for performance measures. It employs a modal logic, PML_μ, which the modeller uses to describe behavioural properties of PEPA models. The states assigned a reward are those which satisfy a behavioural property. PML_μ was shown to be especially suitable for working with PEPA, since it characterises strong equivalence. This was shown by a similar construction to Larsen and Skou [56] for *probabilistic bisimulation*. Two PEPA models are strongly equivalent if and only if they satisfy the same set of PML_μ formulas. The PEPA Reward language also allows the modeller to exploit the structure of a PEPA model by focusing attention on model subcomponents within a larger context. A precise meaning was given to a PEPA context, intuitively a process algebra 'skeleton' with 'holes' in which model subcomponents can be placed. A PML_μ formula is satisfied in context if the set of subcomponents in the view of the context satisfy the formula, subject to the observations that the process may also evolve silently due to subcomponents not in view, and further that the timing behaviour specified by the formula is satisfied by the model, not the subcomponents. Conditions were then given for the preservation of the truth of a PML$_\mu$ formula when aggregating in context. In essence, these conditions are that any context representing a component to be aggregated must be such that all its subcomponents are within the view of the context. These results ensure the PEPA modeller may specify a reward structure by studying the behaviour of a set of subcomponents of a model in a larger context, and yet may still aggregate particular subcomponents with a guarantee that performance measures will be preserved.

A new model construction technique was presented, the aim of which is to allow the use of generally distributed activities in PEPA models. The technique provides a new derived PEPA combinator. The combinator arguments are sequential components, that is PEPA models that do not consist of a cooperation of further subcomponents, and sets of action types and rates. This combinator implicitly introduces an *arbiter* subcomponent, which is placed in cooperation with the sequential components. Its effect is to force subcomponents into a bottleneck at particular points in their evolution, such that they may have to queue in order to proceed. Subcomponents must leave the queue at a rate dependent on the length of the queue. This structure of model was mapped to a GSMP, and by deducing the general solution form of such models, insensitivity theory was applied to conclude that a particular set of the model's activities need not be assumed to be exponentially distributed, while crucially retaining tractability of solution. Interestingly, the general solution form of such models is a product form over the subcomponents.

The thesis then revisited insensitivity, without recourse to the GSMP. An extension of PEPA called gPEPA was introduced, a syntax for describing models with generally distributed activities. Some restrictions were placed on gPEPA models, for example that generally distributed activities could not cooperate, and general distributions were then modelled using Erlang mixtures. This construction meant that the evolution of the model could be described using only probabilistic and exponentially distributed transitions, which was necessary for the balance equation analysis presented. An operational semantics was presented, partly in terms of probabilistic transitions, and the performance model was shown to be an infinite-state continuous time Markov chain. The main result was that if model subcomponents exhibit a local balance property, then again, the steady-state solution of the model is provably equal to that for a corresponding Markovian version. This recovers the essence of insensitivity. The difference to the work presented earlier is that models need not conform to the structure enforced by a particular combinator. However, conditions on the structure of insensitive process algebras models were not provided; they were presented as local balance conditions, at the level of the stochastic process.

To demonstrate the applicability of these techniques, a case study was presented. PEPA models of transaction processing systems were produced. The new combinator was demonstrated to satisfactorily model one class of database models. The PEPA Reward language was then used to automatically generate reward structures which duplicate performance measures presented in the literature. The purpose of the case study was to exemplify the use of the techniques, rather than to draw any conclusions of novel interest to those in the field of database systems.

7.2 Topics for Further Work

The PEPA Reward language is new, and its utility remains to be demonstrated. Certainly it is the most expressive technique available for specifying performance measures of SPA models. One area of concern is the exposure of the PEPA modeller to unnecessary theory. With expressiveness comes additional complexity, and though straightforward to those familiar with the use of modal logics, PML_μ might be daunting to the user. This could cause the user to disregard the technique, or to use it improperly, leading to mistakes. It would seem useful to provide a 'sugar' for the theory, which may even be used to restrict the modeller to useful subsets of the language. An interesting avenue would be to translate other techniques, such as EMPA_r [5] into the Reward language.

At the end of Chapter 3, a conjecture is made on the use of rate-reduced PML_μ formulas in reward specifications. Proving this conjecture could allow the modeller to aggregate more subcomponents, while still preserving the integrity of existing performance measures. The ability of the Reward language to study subcomponents in context suggests links to a 'weak' version of the modal logic PML_μ. A weak modal formula, such as $\langle\!\langle\alpha\rangle\!\rangle F$, is satisfied if an α is possible during some finite (possibly zero) length sequence of unobservable τ actions, leading to a derivative which satisfies F. The semantics of a PML_μ formula in context are similar, in that subcomponents not in the view of the context are free to perform activities. This link is worth exploring, and may provide the modeller with additional insight into the use of the Reward language.

A more down-to-earth concern is the implementation of the Reward language. Certainly the technique should be implemented in its entirety, within the PEPA Workbench. Tool support should allow the detection of reward preserving aggregations within a context, and warn the user if aggregation will impact upon existing reward specifications. Furthermore, the calculation of performance measures should be efficient. A reasonable naïve runtime can be given as $O(m \times 2^n)$, where the PEPA model has m states, and the n is the maximum depth of any PML$_\mu$ subformula in the reward specification (since the logic possesses no temporal combinators). It may be possible to improve this.

The new combinator for building insensitive models provides interesting links to another area of research in Markovian process algebras, that of identifying structures for product form solution. The nature of the combinator is that it implicitly introduces queues as interacting components into a model of independent subcomponents. In queueing theory, the BCMP theorem [4] states that a queueing model with nodes in any of four classes of service centre, and with multi-class traffic, has a product form solution for the steady-state distribution of the customers at the nodes. The new combinator produces models with a product form solution; moreover, the queues introduced by the combinator may be viewed as first-come-first-served nodes, and the independent activities of the subcomponents may be viewed as infinite-server nodes, two types of node covered by the BCMP theorem. It may be that the class of models presented in Chapter 4 may be extended with PEPA analogues of processor-sharing and last-come-first-served preemptive-resume nodes, while retaining the product form solution.

Other classes of SPA models with a product form solution have been identified. For example, in [46], Hillston and Thomas demonstrate a class of PEPA models which satisfy a set of exclusion conditions identified by Boucherie [10]. These models are similar in construction to those presented in Chapter 4. They consist of a 'resource' interacting with the independent parallel composition of a set of processes. In line with Boucherie's work, a restriction on these models is that if a process P holds the resource, and another, Q, needs it now or at any point in the future, Q is blocked. These models have a product form solution over submodels. An interesting piece of future work will be to attempt to combine these two approaches to product form. It may be the case that Boucherie style resource access, and the interaction imposed by the new combinator, can be combined to produce a more general structure of PEPA model which exhibits a product form solution.

The insensitivity results presented in Chapter 5 present further balance equation conditions on PEPA models. However, there are still process algebra models, which are insensitive to the distributions of some activities, which fall outside of the frameworks presented in this thesis. For example, recall the model presented in Section 4.4.

$$
\begin{aligned}
P &\stackrel{\text{def}}{=} (\alpha, r).P' \\
Q &\stackrel{\text{def}}{=} (\alpha, s).Q' \\
R &\stackrel{\text{def}}{=} P \underset{\{\alpha\}}{\bowtie} P \underset{\{\alpha\}}{\bowtie} Q \underset{\{\alpha\}}{\bowtie} Q
\end{aligned}
\tag{7.2.1}
$$

Process R is a cooperation between four subcomponents, but each must cooperate with the others to perform an activity of type α. At the derivation graph level, this one transition represents the evolution of four subcomponents. At the stochastic process level, a change of state due to α is only represented once. It is simple to see that as long as the original mean is maintained, the distribution of the duration of this activity may be arbitrary without affecting the steady-state solution. Future work could attempt to identify more classes of process algebra structure which guarantee the insensitivity of activity durations.

Insensitivity gives great flexibility, but the conditions required to guarantee the property are strict. It is expected that naturally occurring insensitive process algebra structures may appear infrequently when large 'real-world' models are being studied. However, results such as those published by Henderson and Lucic [38] in the context of stochastic Petri nets suggest that it may be possible to aggregate SPA models to produce an insensitive 'skeleton', and then disaggregate exactly to generate the complete steady-state distribution for the model. It is the author's belief that the structure inherent in SPA models will assist in identifying just such an insensitive skeleton.

Bibliography

[1] A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton. Verifying continuous-time markov chains. In Rajeev Alur and Thomas A. Henzinger, editors, *Eighth International Conference on Computer Aided Verification CAV*, volume 1102 of *Lecture Notes in Computer Science*, pages 269–276, New Brunswick, NJ, USA, July/August 1996. Springer Verlag.

[2] M. Ajmone Marsan and C. Chiola. On Petri nets with deterministic and exponentially distributed firing times. *Lecture Notes in Computer Science "Advances in Petri Nets", Springer-Verlag*, 266:132–145, 1987.

[3] C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, (11):125–155, 1998.

[4] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, April 1975.

[5] M. Bernardo. An algebra-based method to associate rewards with EMPA terms. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Automata, Languages and Programming, 24th International Colloquium*, volume 1256 of *Lecture Notes in Computer Science*, pages 358–368, Bologna, Italy, 7–11 July 1997. Springer-Verlag.

[6] M. Bernardo. *Theory and Application of Extended Markovian Process Algebra*. PhD thesis, University of Bologna, Italy, 1999.

[7] M. Bernardo, W. R. Cleaveland, S. T. Sims, and W. J. Stewart. TwoTowers: A Tool Integrating Functional and Performance Analysis of Concurrent Systems. In *Proc. of the IFIP Joint Int. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing, and Verification (FORTE/PSTV)*, pages 457–467, 1998.

[8] M. Bernardo and R. Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202(1–2):1–54, 28 July 1998.

[9] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems, North-Holland*, 14:25–59, 1987.

[10] R.J. Boucherie. A Characterisation of Independence for Competing Markov Chains with Applications to Stochastic Petri Nets. *IEEE Transactions on Software Engineering*, 20(7):536–544, July 1994.

[11] H. Bowman, J. Bryans, and J. Derrick. Analysis of a multimedia stream using stochastic process algebra. In Corrado Priami, editor, *Sixth International Workshop on Process Algebras and Performance Modelling*, pages 51–69, Nice, September 1998.

[12] M. Bravetti, M. Bernardo, and R. Gorrieri. Towards performance evaluation with general distributions in process algebras. *Lecture Notes in Computer Science*, 1466:405–422, 1998.

[13] M. Bravetti and R. Gorrieri. Interactive Generalized Semi-Markov Processes. In J. Hillston and M. Silva, editors, *Proceedings of the 7th Workshop on Process Algebras and Performance Modelling*, pages 83–98. Prensas Universitarias de Zaragoza, September 1999.

[14] G. Ciardo, R. German, and C. Lindemann. A characterization of the stochastic process underlying a stochastic Petri net. *IEEE Transactions on Software Engineering*, 20(7):506–515, July 1994.

[15] G. Clark. Formalising the Specification of Rewards with PEPA. In M. Ribaudo, editor, *Proceedings of the Fourth Process Algebras and Performance Modelling Workshop*, pages 139–160, July 1996.

[16] G. Clark. Stochastic Process Algebra Structure for Insensitivity. In J. Hillston and M. Silva, editors, *Proceedings of the 7th Workshop on Process Algebras and Performance Modelling*, pages 63–82. Prensas Universitarias de Zaragoza, September 1999.

[17] G. Clark, S. Gilmore, and J. Hillston. Specifying Performance Measures for PEPA. In J. P. Katoen, editor, *Proceedings of 6th AMAST Workshop on Probabilistic and Real-Time Systems, ARTS'99*, volume 1601 of *Lecture Notes in Computer Science*, pages 211–227. Springer-Verlag, May 1999.

[18] G. Clark, S. Gilmore, J. Hillston, and N. Thomas. Experiences with the PEPA Performance Modelling Tools. *IEE Proceedings—Software*, 146(1):11–19, February 1999. Special issue of papers from the Fourteenth UK Performance Engineering Workshop.

[19] G. Clark and J. Hillston. Towards automatic derivation of performance measures from PEPA models. In R. Pooley and J. Hillston, editors, *Proceedings of the Twelfth UK Performance Engineering Workshop*, pages 65–81, September 1996.

[20] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems*, 8:244–263, 1986.

[21] P. R. D'Argenio, J.-P. Katoen, and E. Brinksma. An algebraic approach to the specification of stochastic systems (extended abstract). In D. Gries and W.-P. de Roever, editors, *Proceedings of the IFIP Working conference on Programming Concepts and Methods, PROCOMET'98,* Shelter Island, New York, USA, IFIP Series, pages 126–147. Chapman & Hall, 1998.

[22] C. J. Date. *An Introduction to Database Systems.* Addison-Wesley, Reading, 1st edition, 1983.

[23] L. De Alfaro and Z. Manna. Verification in Continuous Time by Discrete Reasoning. *Lecture Notes in Computer Science*, 936:292–, 1995.

[24] N. M. van Dijk. *Queueing Networks and Product Forms: A Systems Appoach.* John Wiley and Sons, Chichester, 1993.

[25] S. Donatelli, J. Hillston, and M. Ribaudo. A Comparison of Performance Evaluation Process Algebra and Generalized Stochastic Petri Nets. In *Proc. of the 6th Petri Nets and Performance Models Workshop*, pages 158–168, October 1995.

[26] J. B. Dugan, K. S. Trivedi, R. Geist, and V. Nicola. Extended stochastic Petri nets: Applications and analysis. Technical Report DUKE–TR–1984–16, Department of Computer Science, Duke University, 1984.

[27] A. El-Rayes, M. Kwiatkowska, and S. Minton. Analysing performance of lift systems in PEPA. In R. Pooley and J. Hillston, editors, *Proceedings of the Twelfth UK Performance Engineering Workshop*, pages 83–100, September 1996.

[28] A. El-Rayes, M. Kwiatkowska, and G. Norman. Solving Infinite Stochastic Process Algebra Models through Matrix Geometric Methods. In J. Hillston and M. Silva,

editors, *Proceedings of the 7th Workshop on Process Algebras and Performance Modelling*, pages 41–62. Prensas Universitarias de Zaragoza, September 1999.

[29] 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.

[30] 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 N. Götz, U. Herzog, and M. Rettelbach, editors, *PAPM '93: Workshop on Formalisms, Principles and State-of-the-Art*, volume 26, pages 89–114, 1993.

[31] J. F. Groote. Transition system specifications with negative premises. *Theoretical Computer Science*, 118(2):263–299, 27 September 1993.

[32] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[33] P. Harrison and J. Hillston. Exploiting quasi-reversible structures in Markovian process algebra models. *The Computer Journal*, 38(7):510–520, 1995.

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

[35] P. G. Harrison and B. Strulo. *Stochastic Process Algebra for Discrete Event Simulation*, pages 18–37. Esprit Basic Research Series. Springer Verlag, 1995.

[36] B. R. Haverkort and K. S. Trivedi. Specification techniques for Markov reward models. *Discrete Event Dynamic Systems: Theory and Applications*, 3(2/3):219–247, July 1993.

[37] W. Henderson and D. Lucic. Applications of Generalised Semi Markov Processes to Stochastic Petri Nets. *Performance of Distributed and Parallel Systems*, pages 315–328, 1989.

[38] W. Henderson and D. Lucic. Aggregation and Disaggregation Through Insensitivity in Stochastic Petri Nets. *Performance Evaluation*, (17):91–114, 1993.

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

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

[41] H. Hermanns. *Interactive Markov Chains.* PhD thesis, University of Erlangen-Nürnberg, Germany, 1998.

[42] H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis, and M. Siegle. Compositional performance modelling with TIPPtool. *Performance Evaluation*, 39:5–35, 2000.

[43] J. Hillston. PEPA: Performance Enhanced Process Algebra. Technical Report CSR-24-93, Department of Computer Science, The University of Edinburgh, 1993.

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

[45] J. Hillston and M. Ribaudo. *Stochastic Process Algebras: A New Approach to Performance Modeling.* Chapter 10, State of the Art Modeling and Simulation of Advanced Computer Systems, 1996.

[46] J. Hillston and N. Thomas. Product Form Solution for a class of PEPA Models. *Performance Evaluation*, 35:171–192, 1999.

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

[48] D. R. W. Holton. A PEPA specification of an industrial production cell. *The Computer Journal*, 38(7):542–551, 1995.

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

[50] K. Irani and H. L. Lin. Queueing network models for concurrent transaction processing in a data base system. In *ACM SIGMOD, Boston*, 1979.

[51] J.-P. Katoen. *Quantitative and Qualitative Extensions of Event Structures.* PhD thesis, Proefschrift Universiteit Twente, Enschede, Netherlands, 1996.

[52] P. J. B. King. *Computer and Communication Systems Performance Modelling.* International Series in Computer Science. Prentice Hall, 1990.

[53] W. H. Kohler, K. C. Wilner, and J. A. Stankovic. An experimental evaluation of locking policies in a centralized testbed database system. In *Proc. ACM SIGMOD Conf.*, page 108, San Jose, CA, May 1983.

[54] K. G. Larsen. *Context-dependent bisimulation between processes.* PhD thesis, Department of Computer Science, University of Edinburgh, 1986.

[55] K. G. Larsen. Compositional theories based on an operational semantics of contexts. In *REX workshop of Stepwise Refinement of Parallel Systems, May 1989*, volume 430 of *LNCS*, pages 487–518. Springer-Verlag, May 1989.

[56] K. G. Larsen and A. Skou. Bisimulation through Probabilistic Testing. *Information and Computation*, 94(1):1–28, September 1991.

[57] C. Lindemann. *Performance Modelling with Deterministic and Stochastic Petri Nets.* John Wiley and Sons, 1998.

[58] K. Matthes. Zur Theorie der Bedienungsprozesse. In *Trans. 3rd Prague Conference Inf. Th., Stat. Dec. Fns. and Rand. Proc.*, pages 513–528, 1962.

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

[60] V. F. Nicola. Lumping in Markov reward processes. *IBM T.J.Watson Res. Center, P.O.Box 704, Yorktown Heights, New York 10598*, 1990.

[61] G. D. Plotkin. A structural approach to operational semantics. Technical Report 19, Åarhus University, 1981.

[62] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C, 2nd. edition.* Cambridge University Press, 1992.

[63] K. H. Pun and G. G. Belford. Performance study of two phase locking in single-site database systems. *IEEE Transactions on Software Engineering (SE), Vol.SE-13*, (12), December 1987.

[64] R. Cleaveland and S. Sims. The NCSU Concurency Workbench. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102 of *Lecture Notes in Computer Science*, pages 394–397, New Brunswick, NJ, USA, July/August 1996. Springer Verlag.

[65] M. Rumsewicz and W. Henderson. Insensitivity with age-dependent routing. *Adv. Appl. Prob., Vol. 21*, pages 398–408, 1989.

[66] W. H. Sanders and J. F. Meyer. A unified approach for specifying measures of performance, dependability, and performability. *Dependable Computing and Fault Tolerant Systems*, 4, 1991.

[67] R. Schassberger. Insensitivity of Steady-State Distributions of Generalized Semi-Markov Processes. *The Annals of Probability. Part I: Vol. 5, :1, 87-99. Part II: Vol. 6 (1978):1, 85-93*, 5, 1:87–99, 1977.

[68] M. Sereno. Towards a Product Form Solution for Stochastic Process Algebras. *The Computer Journal*, 38(7):622–632, 1995.

[69] G. Shedler. *Regenerative Stochastic Simulation*. Statistical Modelling and Decision Science. Academic Press Inc., 1993.

[70] R. M. Smith and K. S. Trivedi. The Analysis of Computer Systems using Markov Reward Processes. *Stochastic Analysis of Computer and Communication Systems*, pages 589–629, 1990.

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

[72] A. Thomasian. Concurrency control: methods, performance, and analysis. *ACM Computing Surveys*, 30(1):70–119, March 1998.