CHAPTER 10

STOCHASTIC PROCESS ALGEBRAS: A NEW APPROACH TO PERFORMANCE MODELING

Jane Hillston and Marina Ribaudo

10.1 INTRODUCTION

In this chapter we present an introduction to the novel approach to performance modeling provided by Stochastic Process Algebras (SPA). Like queueing networks and stochastic Petri nets, and their variants, these formal languages can be regarded as high-level model specification languages for low-level stochastic models. The advantages of SPAs are that they incorporate the attractive features of process algebras and thus bring to the area of performance modeling several attributes which are not offered by the existing formalisms. Perhaps the most important such feature is the compositionality which is inherent in the models and can be exploited during their analysis. Throughout this chapter we will present several simple examples which illustrate this and several other aspects of the SPA approach to performance modeling.

In recent years several SPA languages have been developed. Here we will focus upon Hillston's Performance Evaluation Process Algebra (PEPA). The other published languages are discussed in Section 10.6.

The rest of the chapter is structured as follows. In Section 10.2 we present a brief overview of process algebras and their recent timed and probabilistic extensions. PEPA is introduced in some detail in Section 10.3, and the associated approach to model construction is discussed.

The analysis of models and the extraction of performance measures are presented in Section 10.4. Facilities for manipulating models, model simplification and aggregation, are outlined in Section 10.5. In particular we discuss the advantages brought to such manipulation by the compositional structure and equivalence relations inherent in SPA. In Section 10.6 we give a summary of the different features of the other SPA languages and in Section 10.7 we outline recent research work which aims to exploit the compositional structure within SPA models to find efficient solution techniques. Finally we summarise the attractive features offered by this new approach.

10.2 CLASSICAL PROCESS ALGEBRA

Process algebras are abstract languages used for the specification and design of concurrent systems. The most widely used process algebras are Milner's Calculus of Communicating Systems (CCS)[20] and Hoare's Communicating Sequential Processes (CSP)[18] and the SPAs take inspiration from both these formalisms. Models in CCS and CSP have been used extensively to establish the correct behavior of complex systems by deriving *qualitative* properties such as *freedom from deadlock* or *livelock*.

In the process algebra approach systems are modeled as collections of entities, called *agents*, which execute atomic *actions*. These actions are the building blocks of the language and they are used to describe sequential behaviors which may run concurrently, and synchronisations or communications between them.

In CCS two agents communicate when one performs an action, a say, while the other performs the complementary action $\overline{a}$. The resulting communication action has the distinguished label τ, which represents an *internal* action that is invisible to the environment. Agents may proceed with their internal actions simultaneously but it is important to note that the semantics given to the language imposes an interleaving on such concurrent behavior. The grammar of the language makes it possible to construct an agent which has a designated first action (prefix); has a choice over alternatives (choice); or has concurrent possibilities (composition).

The communication mechanism is different in CSP as there is no notion of complementary actions: this is a major distinction between CCS and CSP. In CSP two agents communicate by simultaneously executing actions with the same label. Since during the communication the joint action remains visible to the environment, it can be reused by other concurrent processes so that more than two processes can be involved in the communication. This is the communication mechanism adopted in the SPA languages.

In CCS and CSP, since the objective is qualitative analysis rather than quantitative, time is abstracted away. In the last decade various suggestions for incorporating time into these formalisms have been investigated (for an overview see 21). However, most of these retain the assumption that actions are instantaneous and regard time progressing as orthogonal to the activity of the system. These assumptions make such models unsuitable for performance analysis. In contrast, SPAs associate a random variable, representing duration, with each action.

Similarly, process algebras are often used to model systems in which there is uncertainty about the behavior of a component, but this uncertainty is also abstracted away so that all choices become nondeterministic. Probabilistic extensions of process algebras, such as PCCS[19], allow this uncertainty to be quantified using a probabilistic choice combinator. In this case a probability is associated with each possible outcome of a choice. In SPA an alternative approach is taken—we assume that a *race condition* resolves choices when more than one action can occur.

10.3 PEPA

Process algebras offer several attractive features which are not necessarily available in existing performance modeling paradigms. The most important of these are *compositionality*, the ability to model a system as the interaction of its subsystems, *formality*, giving a precise meaning to all terms in the language, and *abstraction*, the ability to build up complex models from detailed components but disregarding internal behavior when it is appropriate to do so. Queueing networks offer compositionality but not formality; stochastic extensions of Petri nets offer formality but not compositionality; neither offer abstraction mechanisms.

Performance Evaluation Process Algebra[14] (PEPA) extends classical process algebra by associating a random variable, representing duration, with every action. These random variables are assumed to be exponentially distributed and this leads to a clear relationship between the process algebra model and a continuous time Markov chain (CTMC). Via this underlying CTMC performance measures can be extracted from the model.

PEPA models are described as interactions of *components*. Each component can perform a set of actions: an action $a \in \mathcal{A}ct$ is described as a pair (α, r), where $\alpha \in \mathcal{A}$ is the *type* of the action and $r \in \mathbb{R}^+$ is the parameter of the negative exponential distribution governing its duration. Whenever a process P can perform an action, an instance of a given probability distribution is sampled: the resulting number specifies how long it will take to *complete* the action. A small but powerful set of combinators is used to build up complex behavior from simpler behavior. The combinators are familiar: sequential composition, selection, synchronisation and encapsulation. We will explain each in its turn below. A formal operational semantics for PEPA is available in 14 and is not reproduced here.

Sequential composition: A component may have purely sequential behavior, repeatedly undertaking one activity after another and eventually returning to the beginning of its behavior. A simple example is a memory module in a multiprocessor system, which allows one data transfer at a time. Each processor requiring memory access will need to acquire the memory and only when the transfer is complete will the memory be released and available again for acquisition.

$$Mem \stackrel{\text{def}}{=} (get, \top).(use, \mu).(rel, \top).Mem$$

In some cases, as here, the rate of an action is outside the control of this component. Such actions are carried out jointly with another component, with this component playing a passive role. For example, the memory is passive with respect to the *get* action and this is recorded by the distinguished symbol, $\top$ (called "top").

Selection: A choice between two possible behaviors is represented as the sum of the possibilities. For example, if we consider a processor in the multiprocessor system, a computation may have two possible outcomes: access to local memory is necessary (with probability p_1) or access to a common memory module is necessary (with probability $p_2 = 1 - p_1$). These alternatives are represented as shown below:

$$\begin{aligned} Proc \stackrel{\text{def}}{=}\; & (think, p_1\lambda).(local, m).Proc \quad + \\ & \quad (think, p_2\lambda).(get, g).(use, \mu).(rel, r).Proc \end{aligned}$$

A race condition is assumed to govern the behavior of simultaneously enabled actions so the choice combinator represents pre-emptive selection with re-sampling. The continuous nature of the probability distributions ensures that the actions cannot occur simultaneously. Thus a sum will behave as either one summand or the other. When an action has more than one possible outcome, e.g. the *think* action in the processor, it is represented by a choice of separate actions, one for each possible outcome. The rates of these actions are chosen to reflect their relative probabilities.

Synchronisation: We have already anticipated that the processor and the memory in the example will be working together within the same system. This will require them to *cooperate* when the processor needs access to data which is not available locally. In contrast, the local activities of the processor can be carried out independently of the memory. Cooperation over given actions is reflected in the cooperation by the *cooperation set*, $L = \{get, use, rel\}$ in this case. Actions in this set require the simultaneous involvement of both components. The resulting action, a *shared* action, will have the same type as the two contributory actions and a rate reflecting the rate of the action in the slowest participating component. Note that this means that the rate of a passive action will become the rate of the action it cooperates with.

If, for simplicity, we assume that the multiprocessor consists of just two processors, the system is represented as the cooperation of the processors and the memory as follows:

$$Sys_1 \stackrel{\mathrm{def}}{=} (Proc \parallel Proc) \underset{L}{\bowtie} Mem \qquad L = \{get, use, rel\}$$

The combinator $\parallel$ is a degenerate form of the cooperation combinator, formed when two components behave completely independently, without any cooperation between them, as in the case of the two processors. This pure parallel combinator can be thought of as cooperation over the empty set: $(Proc \underset{\emptyset}{\bowtie} Proc)$.

Encapsulation: It is often convenient to hide some actions, making them private to the component or components involved. The duration of the actions is unaffected, but their type becomes hidden, appearing instead as the unknown type τ. Components cannot synchronise on τ. For example, as we further develop the model of the multiprocessor we may wish to hide the access of a processor to its local memory. This might lead to a new representation of the processor:

$$Proc' \stackrel{\text{def}}{=} Proc/\{local\}$$

and a corresponding new representation of the system:

$$Sys_2 \stackrel{\text{def}}{=} (Proc' \parallel Proc') \underset{L}{\bowtie} Mem \qquad L = \{get, use, rel\}$$

Use of the hiding combinator in this way has two implications. Firstly, it ensures that no components added to the model at a later stage can interact, or interfere, with this action of the processor. Secondly, as we will see in Section 10.4., private actions are deemed to have no contribution to the performance measures being calculated and this might subsequently suggest simplifications to the model.

Throughout the simple example above we have used constants such as Mem to associate names with behaviors. Using recursive definitions we have been able to describe components with infinite behaviors without the use of an explicit recursion operator.

Representing the components of the system as separate components means that we can easily extend our model. Now we may want to consider a system consisting of more than two processors which act independently of each other but compete for the use of common memory. This extension may be achieved compositionally by combining more instances of the $Proc$ component already described. For example, in the case of four processors we have:

$$Sys_3 \stackrel{\text{def}}{=} (Proc \parallel Proc \parallel Proc \parallel Proc) \underset{L}{\bowtie} Mem \qquad L = \{get, use, rel\}$$

10.4 MODEL ANALYSIS

The formality of the process algebra approach allows us to assign a precise meaning to every language expression. This implies that once we have a language description of a given system its behavior can be deduced automatically. The meaning, or semantics, of a PEPA expression is provided by the formal semantics, in the structured operational style, which associates a labeled multi-transition system with every expression in the language[14].

A labeled transition system $(S, T, \{\xrightarrow{t} \mid t \in T\})$ consists of a set of states S, a set of transition labels T and a transition relation

$\xrightarrow{t} \subseteq S \times S$. For PEPA the states are the syntactic terms in the language, the transition labels are the actions ($(type, rate)$ pairs), and the transition relation is given by the semantic rules. A multi-transition relation is used because the number of instances of a transition (action) is significant since it can affect the timing behavior of a component.

Based on the transition relation, a transition diagram, called the *derivation graph* (DG), can be associated with each language expression. This graph describes all the possible evolutions of any component and provides a useful way to reason about the behavior of a model. An example derivation graph is shown in Figure 10.1 where the DG of the PEPA model Sys_4, consisting of a single processor accessing the global memory, is shown. For simplicity, in the upper part of the figure we have expanded the derivatives of the components $Proc$ and Mem.

$$
\begin{array}{rcllll}
Proc & \stackrel{\text{def}}{=} & (think, p_1\lambda).Proc_1 + (think, p_2\lambda).Proc_2 & & & \\
Proc_1 & \stackrel{\text{def}}{=} & (local, m).Proc & & & \\
Proc_2 & \stackrel{\text{def}}{=} & (get, g).Proc_3 & Proc_3 \stackrel{\text{def}}{=} (use, \mu).Proc_4 & & Proc_4 \stackrel{\text{def}}{=} (rel, r).Proc \\
Mem & \stackrel{\text{def}}{=} & (get, \top).Mem_1 & Mem_1 \stackrel{\text{def}}{=} (use, \mu).Mem_2 & & Mem_2 \stackrel{\text{def}}{=} (rel, \top).Mem \\
Sys_4 & \stackrel{\text{def}}{=} & Proc \underset{L}{\bowtie} Mem & L = \{get, rel, use\} & &
\end{array}
$$

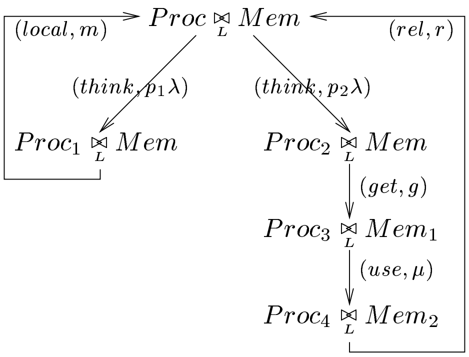

Figure 10.1: Derivation graph underlying Sys_4.

Inspection of the DG allows one to derive qualitative properties of the model. In this case, for instance, we can see that the PEPA model is free from deadlock and live. Moreover, the CTMC underlying any finite PEPA component can be obtained directly from the DG: a state of the Markov process is associated with each node of the graph and the transitions between states are defined by considering the rates labeling the arcs. Since all activity durations are exponentially distributed, the total transition rate between two states will be the sum of the activity rates labeling arcs connecting the corresponding nodes in the DG. Starting from the DG of Figure 10.1, the derivation of the corresponding CTMC is straightforward and results in the generator matrix shown below.

$$\boldsymbol{Q} = \begin{pmatrix} -\lambda & p_1\lambda & p_2\lambda & 0 & 0 \\ m & -m & 0 & 0 & 0 \\ 0 & 0 & -g & g & 0 \\ 0 & 0 & 0 & -\mu & \mu \\ r & 0 & 0 & 0 & -r \end{pmatrix}$$

In order to ensure that the underlying Markov process is ergodic, the DG of a PEPA model must be strongly connected. Necessary conditions for ergodicity, at the syntactic level of a PEPA model, have been defined in 14. For example, if cooperation occurs it must be the highest level combinator. The class of PEPA terms which satisfy these syntactic conditions are termed *cyclic components* and they can be described by the following grammar:

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

All the models we have discussed so far are cyclic models.

It is well known that if the CTMC is ergodic, it is possible to compute the steady state probability distribution over all the possible states by solving the matrix equation $\pi\boldsymbol{Q} = 0$ where $\boldsymbol{Q}$ is the state transition rate matrix of the CTMC and π is the state probability vector, such that $\sum_i \pi_i = 1$.

The probability distribution of the states of the model is often not the ultimate goal of performance analysis. In order to define performance measures such as throughput and utilisation the modeler can define a reward structure over the CTMC at the level of the process algebra. This is done by associating rewards with actions: the reward associated with a state is then the total reward attached to the actions it enables. Note that no reward can be attached to internal, τ, actions.

We now describe in some detail a modified version of the previous example showing how we can extract performance measures from the model. We consider again a multiprocessor system with a shared memory in which all processes have the same functional behavior: they compete for access to the shared memory, use and then release it. In this case, however, we assume that basic actions progress at different speeds depending on the processor on which they are running, and that the action *think* models both the processing activity and the access to the local memory. Thus a process running on the ith processor has the following specification:

$$P_i \stackrel{\text{def}}{=} (think, \lambda_i).(get, g).(use, \mu_i).(rel, r).P_i$$

When n_i processes P_i are running on the ith processor the system is modeled by considering n_i independent replicas of the same process:

$$Sys_5 \stackrel{\text{def}}{=} (\underbrace{P_1 \| \cdots \| P_1}_{n_1} \| \cdots \| \underbrace{P_N \| \cdots \| P_N}_{n_N}) \underset{L}{\bowtie} Mem \qquad L = \{get, use, rel\}$$

The DG and the CTMC underlying the model can automatically be obtained using the PEPA Workbench[7] when we instantiate the model parameters, i.e. the number of processors and the number of processes. In the simple case of $N = 3$, $n_1 = n_2 = 2$, $n_3 = 1$ we obtain

$$Sys_6 \stackrel{\text{def}}{=} (P_1 \parallel P_1 \parallel P_2 \parallel P_2 \parallel P_3) \underset{L}{\bowtie} Mem \qquad L = \{get, use, rel\}$$

whose DG has 192 states.

The CTMC is derived directly from the DG by considering only the action rates labeling the arcs. By solving the set of linear equations we obtain the steady state probability vector. If we instantiate the rate parameters of the model as follows,

$$r = g = 100,\ \lambda_1 = 3,\ \lambda_2 = 2,\ \lambda_3 = 3,\ \mu_1 = 0.5 \times m,\ \mu_2 = \mu_3 = 3$$

where m takes values in the range $1, \ldots, 5$, and calculate, for each processor, the percentage of time it spends waiting for access to the memory relative to the time it spends accessing the memory, we obtain the graph shown in Figure 10.2.

The percentage waiting time for each processor is found by associating a reward of 1 with each derivative in which *get* is the only action enabled by any process on that processor, but the memory is not enabling *get*. The time accessing the memory is found by associating a reward 1 with each derivative in which any process is enabling *use* or *rel*, or enabling *get* when the memory is also enabling *get*. The graph suggests that the processes running on Processor 1 are monopolising the memory.

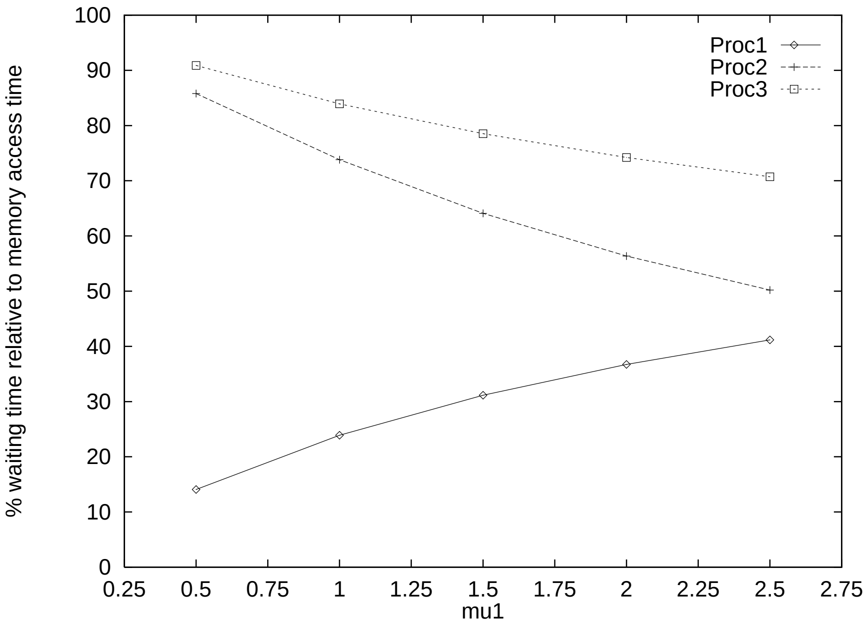

Figure 10.2: Percentage waiting times for Processors 1, 2 and 3 as the mean duration of Processor 1 memory access decreases

One way in which we might alleviate this problem would be to introduce a protocol for memory access which attempts to restrict the patterns of behavior exhibited by the processors. In the protocol, represented in the modified model shown below, after granting access to a given processor the memory will not grant access to that processor again until a different processor has been served. In the component representing the memory we now distinguish the *get* actions originating on different processors, in order to enforce the protocol. The memory also remembers which processor last had access in order to disallow it in the next access. Thus Mem_i represents the memory when it was last accessed by the ith processor and it is modeled as the sum of all its possible behaviors:

$$Mem_i \stackrel{\text{def}}{=} \sum_{\substack{j=1 \\ j \neq i}}^{N} (get_j, \top).(use, \mu_j).(rel, \top).Mem_j$$

Each process model is also modified by renaming its *get* action to get_i, if it is running on Processor i; in all other aspects it remains the same.

The model of a multiprocessor system with this memory access pro-

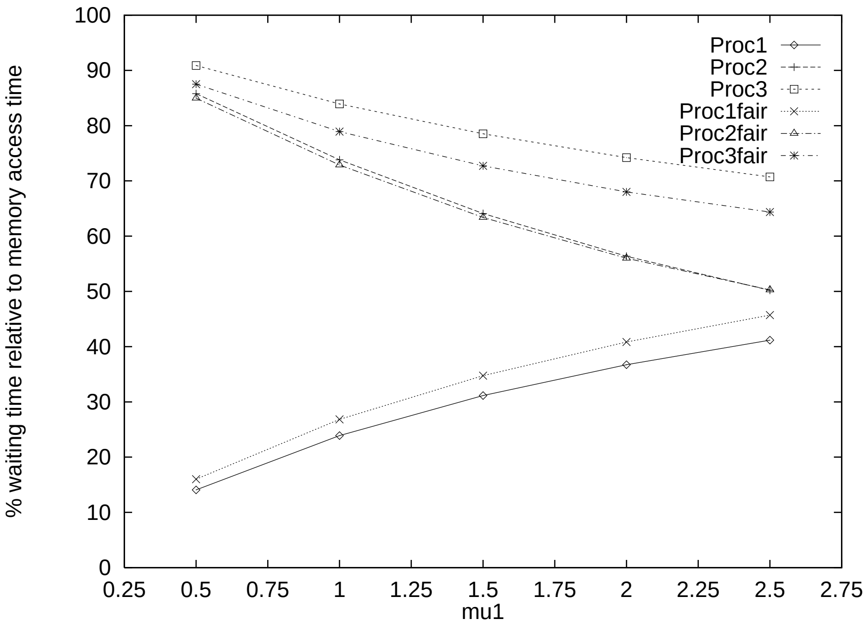

Figure 10.3: Percentage waiting times for Processors 1, 2 and 3 under both access protocols, as the mean duration of Processor 1 memory access decreases

tocol is given below.

$$Sys'_5 \stackrel{\text{def}}{=} (\underbrace{P_1 \parallel \cdots \parallel P_1}_{n_1} \parallel \cdots \parallel \underbrace{P_N \parallel \cdots \parallel P_N}_{n_N}) \underset{L}{\bowtie} Mem_i \qquad L = \{get_i, use, rel\}$$

Note that with the inclusion of Mem_i in the initial system we are imposing that the starting state excludes access of an arbitrary processor $Proc_i$. When, as before, $N = 3$, $n_1 = n_2 = 2$, $n_3 = 1$ we have:

$$Sys'_6 \stackrel{\text{def}}{=} (P_1 \parallel P_1 \parallel P_2 \parallel P_2 \parallel P_3) \underset{L}{\bowtie} Mem_1 \qquad L = \{get_1, get_2, get_3, rel, use\}$$

whose DG has 256 states. As previously, we calculate the percentage time each processor spends waiting for access relative to access time. The results, under both the original access protocol and the new fair protocol, are shown in Figure 10.3. As hoped, this modification to the system does reduce the percentage of time that Processors 2 and 3 spend waiting for access to the memory.

10.5 EQUIVALENCE RELATIONS

As we discussed in Section 10.3, one of the most attractive features of process algebras (both classical and stochastic) is their compositional nature: complex systems can be modeled starting from smaller components considered individually. Each component is developed in detail and subsequent interactions between them are represented.

Another important aspect of the formalism is the definition of equivalence relations[20], which can be used to compare agents (*model verification*) and to replace one agent by another which exhibits an equivalent behavior, but has a simpler representation (*model simplification*). Such notions of equivalence are considered part of the semantics of a process algebra, and therefore their definition is an integral part of the language development.

In CCS for example, two agents are considered to be equivalent when their externally observed behaviors appear to be the same or, in other words, when any action by one can be matched by an action of the other and the resulting derivatives are also equivalent. This is a formally defined notion of equivalence, based on the labeled transition system underlying the process algebra. Different equivalence relations have been proposed depending on whether the action τ is considered to be observable (*strong bisimulation*) or not (*weak bisimulation*)[20].

Such equivalence relations are particularly useful when they can be shown to be a *congruence* and therefore complementary to the structure of the model. A relation is a congruence if it is preserved by all the combinators of the language and this will mean that model verification and model simplification can be carried out on the components within a model, rather than across the whole model at once. If an algebraic characterisation (axiomatisation) is found, the axioms may be used to apply the equivalence relation at the level of the syntax rather than at the level of the underlying transition system.

By extending the standard theoretical results from classical to stochastic process algebras, different notions of equivalence have been defined. Here both functional and temporal aspects have to be taken into account, that is, both actions and delays have to be considered. In this context equivalence relations have been shown to have practical implications for performance modeling.

Currently in performance analysis notions of equivalence are used informally to address the problems of large Markov chains in two important ways. Firstly, an alternative model may be proposed; this could be easier to solve because it is more compact or exhibits special structure which is susceptible to efficient solution, e.g. product form. In this case a notion of equivalence *between models* may be used to establish whether or not the alternative model is an adequate representation of the original (*model simplification*). Secondly, equivalence *within a model* may identify states which exhibit sufficiently similar behavior that they need not be distinguished, thus reducing the size of the model to be solved (*aggregation*).

In queueing networks and stochastic Petri nets there has been little formal development of equivalence between models. In contrast there has been much work on the definition of equivalences at the state space level. These equivalences form the basis of aggregation techniques for reducing the state space of the underlying Markov chain, and thus provide a technique for making the solution of large models tractable.

Stochastic process algebras are the first performance modeling paradigm to offer a formalisation of both notions of equivalence. In the following subsections we discuss two distinct equivalence relations which have been developed for PEPA[14]. These can be used, respectively, for model simplification and aggregation.

10.5.1 Weak Isomorphism

The use of the hiding operator during the development of a PEPA model allows the modeler to represent aspects of a component's behavior in detail but to later abstract away from it. Such restriction may also be used to safeguard the interaction between components from external interference, as we saw in Section 10.3. This style of modeling may mean that, when complete, a model executes sequences of τ, or internal, actions. Since rewards cannot be attached to such actions no performance information is obtained from these sequences. It is natural therefore to try to simplify them.

The most straightforward means of simplifying a sequence of hidden actions is to replace them by a single hidden action, thus amalgamating intermediate states. Such an approach would, in general, lead to an approximation technique, and has not yet been developed. However an equivalence relation called *weak isomorphism* has been developed for PEPA which can be used to identify when such an approach will achieve exact model simplification. Using weak isomorphism the observation of internal activities is relaxed, and insensitive sequences of τ actions may be replaced by a single τ action with appropriate mean.

The calculation of different performance measures may require different action names to be visible or hidden. We can think of this as defining an experimental context for the model. Via judicious use of the PEPA abstraction mechanisms, weak isomorphism allows a model to be modified to a simpler form, reflecting its current experimental context. Although the weak isomorphism relation is not a congruence relation, it has been shown to be preserved by cooperation, and so this model simplification technique can be applied compositionally in cyclic models.

10.5.2 Strong Equivalence

Strong equivalence is a bisimulation which aims to capture when two PEPA components are indistinguishable under experimentation. This equivalence notion considers both the functional and the temporal aspects of the model but observation occurs without detailed knowledge of the individual actions involved. Thus strong equivalence is unable to distinguish between a single $(\alpha, 2r)$ activity and two simultaneously enabled (α, r) activities.

The concept of *conditional transition rate*, $q(C_i, C_j, \alpha)$, is the basis for the definition of strong equivalence. This is the rate at which a system behaving as component C_i evolves to behaving as component C_j as a result of completing an activity of type α. If we consider a *set* of possible derivatives S we can compute the *total conditional transition rate* from C_i to S as the sum of the conditional transition rates from C_i to any component C_j in the set S

$$q[C_i, S, \alpha] = \sum_{C_j \in S} q(C_i, C_j, \alpha)$$

Two PEPA components are considered strongly equivalent if there is an equivalence relation between them such that, for any action type α, the total conditional transition rates from those components to any equivalence class, via activities of this type, are the same.

The formal definition of strong equivalence may be found in 14 where it has been shown that this relation is sufficient to ensure that the components exhibit the same behavior. Moreover, strong equivalence may also be used over the state space underlying a single PEPA model, resulting in a lumpable partition of the state space. The DG of the model is partitioned into equivalence classes and each equivalence class forms a single state in the aggregated Markov process.

Thus the well-known technique of aggregation based on lumpability is now defined at the level of the modeling paradigm rather than of the underlying CTMC. However this implies that the entire state space must be generated. This *a priori* generation of the state space is undesirable and can be avoided because the strong equivalence relation is a congruence.

A procedure for implementing strong equivalence aggregation during the composition of cooperating components is outlined in 16. Partial state spaces (corresponding to cooperating components) are generated and aggregated. Then each pair of cooperating components is replaced by a corresponding aggregated model and the procedure is repeated. Thus the state space of the complete model does not need to be generated. Moreover the formality of the approach has important implications for tool support for model simplification.

We now show how strong equivalence may be used to reduce the state space of the Markov process underlying a PEPA model. We consider again the multiprocessor system with the memory access protocol we discussed in Section 10.4:

$$Sys_6' \stackrel{\text{def}}{=} (P_1 \parallel P_1 \parallel P_2 \parallel P_2 \parallel P_3) \underset{L}{\bowtie} Mem_1 \qquad L = \{get_1, get_2, get_3, rel, use\}$$

Recall that the DG of Sys_6' has 256 states. By applying strong equivalence we can recognise that some states belong to the same equivalence class. For example, if one of the processes on Processor 1 is thinking and the other is using the memory, the behavior of the system is the same as if their roles were reversed, i.e. the first is using the memory and the other is thinking. Exploiting such symmetries between processes running on the same processor we can reduce the state space to 120 states. Note that this is an exact aggregation and the performance measures extracted from the model are unaffected. This corresponds to lumping together states which differ only in a permutation of the derivatives of the components modeling processes running on the same processor. This is shown in diagrammatic form in Figure 10.4, for the case of the two processes P_1. The two states numbered #1 and #2 are strongly equivalent and therefore they belong to the same equivalence class. Similarly for the two states #3 and #4.

Notice that the reductions we have obtained on the example are based strictly on the symmetries of the model which occur because of

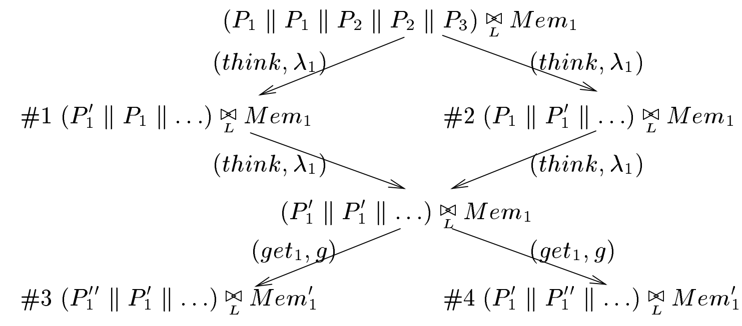

Figure 10.4: Small portion of the derivation graph of Sys_6'.

repeated components. The power of strong equivalence is not limited to this case. For example, consider a model in which the get_i actions are hidden and considered internal to the model. If it is also the case that processes running on the different processors share the same characteristics, i.e. $\lambda_1 = \lambda_2 = \lambda_3$ and $\mu_1 = \mu_2 = \mu_3$, then we are able to obtain a further reduction by lumping more states. This further reduction can be explained by the observation that only one process accesses the memory at a time and it does not really matter which: other processes are indistinguishable. Once the get_i actions become τ actions, and therefore indistinguishable, strong equivalence will result in a further substantial reduction of the state space, to 67 states.

In general, strong equivalence can also be used to find more general forms of repeated patterns of behavior within models. When repeated patterns of behavior are recognised, the states that have the same behavior (i.e. states that after performing the same actions with the same rates reach equivalent states) are lumped together.

10.6 OTHER SPA

Work on SPA originated with Herzog's group at the University of Erlangen. The initial work was carried out on a process algebra EXL which was a variant of CSP in which a random variable was associated with each event and a probabilistic choice operator replaces nondeterministic choice[13]. This language later evolved into TIPP (TImed Process for Performance Evaluation)[8,9]. The work on TIPP has been primarily motivated by a desire to encourage the timely consideration of performance characteristics during system development, particularly for distributed systems[13]. Although originally TIPP was studied under very general assumptions that allowed any distribution to be used to characterise the duration of an action, difficulties in extracting performance measures from such models has meant that more recent work has focussed on a Markovian version of the language (MTIPP)[11] which is very close to PEPA.

Markovian Process Algebra (MPA) was developed by Buchholz at the University of Dortmund[4,5] in 1994. The major difference between this language and PEPA is the assumption that all actions of type α proceed at a fixed rate μ_α. Activities are still represented as (α, r) pairs but r now represents the number of concurrently enabled instances of action α, all of which proceed at rate μ_α. This difference has a subtle impact on the semantics of the language.

Extended Markovian Process Algebra (EMPA) was also developed in 1994 by Gorrieri's group at the University of Bologna[1,2]. Originally this SPA was also called MPA but the name was later changed to avoid ambiguity. The language includes a rich set of combinators. It was the first SPA to include *immediate* or instantaneous actions. Passive actions play a central role in the theory of EMPA and synchronisation is restricted to involve at most one timed or immediate action. Most of the work on EMPA has focussed on Petri net based semantics for the language[3].

The most significant difference between the SPA languages lies in their different definitions of synchronisation. This is discussed in some detail in 15.

10.7 EFFICIENT SOLUTION

SPA models are prone to the problem of state space explosion: even moderately sized models can generate a huge state space making analysis of the system infeasible. Much of the recent work, like the earlier work on equivalence relations, is motivated by a desire to address this problem. Two main strands are identifiable in the current work. The first seeks to characterise models in which the compositional structure may be exploited in order to gain efficient solution of the underlying CTMC by considering submodels in isolation. The second aims to avoid the construction of large CTMCs by removing the condition that each syntactic term encountered in the model is treated as a distinct state.

Given the clear component-based structure of SPA models it is perhaps natural to consider decompositional solution techniques. A class of SPA models susceptible to such a solution technique is presented in 17. Syntactic conditions to identify models which satisfy Courtois's *near complete decomposability* property[6] are presented, meaning that time scale decomposition can be carried out automatically without the need to ever construct and store the whole state space.

Efficient product form solution is one of the major attractions of queueing networks for performance modeling purposes. These models rely on a form of interaction between nodes in a network which allows them to be solved in isolation, since they behave as if independent, up to normalisation. In a recent paper[10], the results from queueing networks have been exploited to identify a restricted form of interaction between suitable SPA components which leads to a product form solution. Another approach, based on work on stochastic Petri nets which satisfy certain structural conditions, has identified another class of product form SPA models[25]. In such models the actions themselves may be considered to be states of a Markov chain which may be solved to find the steady state of the complete process.

Incorporating immediate actions into a model means that "states" enabling such actions can be eliminated. In 12, MTIPP is enriched with immediate actions which allow control activities to be modeled as actions which take a negligible amount of time. In order to derive the underlying CTMC, immediate actions must be removed. This procedure, based on a relation called Markovian observational congruence[12], is only possible for immediate actions which are internal, i.e. ones which do not participate in external synchronisation because they are hidden from the environment. The congruence property ensures that the elimination can be performed on the fly, during the construction of the CTMC.

Immediate actions can also be used to model branching probabilities (as in Generalized Stochastic Petri Nets). In 23 this use of immediate actions is proposed for MTIPP, via the introduction of a distinct operator to represent probabilistic branching. Again, a specific congruence relation is the key to correct derivation of the underlying CTMC.

An alternative compositional approach to the solution of Markov processes is the use of tensor algebra to express the generator matrix of a process[22]. This approach is based on a restricted form of interaction between subsystems within the system and it has also been applied to SPA[4,24].

10.8 CONCLUSIONS

In this chapter we have described an algebraic description technique, based on a classical process algebra, and enhanced with timing information. This extension results in models which may be used to calculate performance measures as well as deduce functional properties of the system. Several interesting features of SPA, such as the compositionality, the abstraction mechanism, and the equivalence notions suitable for performance analysis, have been discussed in some detail. This approach to performance modeling is a recent innovation and a lot of research activity is presently going on as we have indicated in Section 10.7. At the same time tool prototypes are under development to exploit the possibilities for automation presented by the SPA approach.

REFERENCES

1. M. BERNARDO, L. DONATIELLO, and R. GORRIERI, "Modeling and Analyzing Concurrent Systems with MPA," Proc. 2^{nd} Process Algebra and Performance Modeling Workshop, (Erlangen, July 1994).

2. M. BERNARDO, L. DONATIELLO, and R. GORRIERI, "MPA: a Stochastic Process Algebra," Technical Report UBLCS-94-10, University of Bologna, Laboratory of Computer Science, (May 1994).

3. M. BERNARDO, L. DONATIELLO, and R. GORRIERI, "Giving a Net Semantics to Markovian Process Algebras," Proc. 6^{th} Int. Workshop on Petri Nets and Performance Models, (Durham, NC, October 1995).

The papers 1,2, and 3 discuss the EMPA language. The first two describe the language; the third provides both an operational and a denotational net semantics for it.

4. P. BUCHHOLZ, "Markovian Process Algebra: Composition and Equivalence," Proc. 2^{nd} Process Algebra and Performance Modeling Workshop, (Erlangen, July 1994).

5. P. BUCHHOLZ, "On a Markovian Process Algebra," Technical Report 500/1994, University of Dortmund, (1994).

The two papers 5 and 6 describe the MPA language with particular emphasis on the compositional generation of the underlying Markov process using tensor algebra.

6. P.J. COURTOIS, "Decomposability: Queueing and Computer Sys-

tem Applications," ACM Series, Academic Press, (New York, 1977).

7. S. GILMORE and J. HILLSTON, "The PEPA workbench: A Tool to Support a Process Algebra Based Approach to Performance Modeling," Proc. 7^{th} Int. Conf. on Modeling Techniques and Tools for Computer Performance Evaluation, (Vienna, 1994).

This paper describes the PEPA workbench, a prototype which provides a set of simple tools to delegate to machine assistance some of the routine tasks in checking PEPA descriptions, generating the underlying CTMC and calculating performance measures.

8. N. GOTZ, U. HERZOG, and M. RETTELBACH, "TIPP - Introduction and Application to Protocol Performance Analysis," Technical Report, University of Erlangen, (March 1993).

9. N. GOTZ, U. HERZOG, and M. RETTELBACH, "Multiprocessor and Distributed System Design: The Integration of Functional Specification and Performance Analysis Using Stochastic Process Algebra," Tutorial Proc. Performance 1993, vol. 729 of LNCS, (Rome, 1994).

The two papers above present the TIPP language. In 9 general distribution functions are allowed and the difficulty of extracting performance measures in this case are discussed. In 10 the distribution of the delays is restricted to be the negative exponential distribution function.

10. P.G. HARRISON and J. HILLSTON, "Exploiting Quasi-reversible Structures in Markovian Process Algebra Models," The Computer Journal, **38**(7), 1995. Special Issue: Proc. 3^{rd} Process Algebra and Performance Modeling Workshop, (Edinburgh, June 1995).

11. H. HERMANNS and M. RETTELBACH, "Syntax, Semantics, Equivalences, and Axioms for MTIPP," Proc. 2^{nd} Process Algebra and Performance Modeling Workshop, (Erlangen, July 1994).

This paper presents a notion of equivalence, called Markovian bisimulation, which is essentially the same as strong equivalence. In addition an equational theory for this equivalence is presented for models with finite (i.e. terminating) behavior. This technique allows model aggregation via syntactic manipulation of the process algebra description (term rewriting).

12. H. HERMANNS, M. RETTELBACH, and T. WEISS, "Formal Characterisation of Immediate Actions in SPA with Nondeterministic Branching," The Computer Journal, **38**(7), 1995. Special Issue: Proc.

3^{rd} Process Algebra and Performance Modeling Workshop, (Edinburgh, June 1995).

13. U. HERZOG, "Formal Description, Time and Performance Analysis: A framework," Technical Report 15/90, IMMD VII, Friedrich-Alexander-Universität, Erlangen-Nürnberg, (Germany, September 1990).

14. J. HILLSTON, "A Compositional Approach to Performance Modeling," PhD thesis CST-107-94, Computer Science Department, University of Edinburgh, (April 1994).

    All the information on the PEPA language, the equivalence relations, and the computation of performance measures may be found in this PhD thesis which will be published by the Cambridge University Press in 1996.

15. J. HILLSTON, "The Nature of Synchronisation," Proc. 2^{nd} Process Algebra and Performance Modeling Workshop, (Erlangen, July 1994).

    This paper compares the different SPA languages with particular emphasis on the various combinators used to model synchronisation between components.

16. J. HILLSTON, "Compositional Markovian Modeling Using a Process Algebra," Proc. 2^{nd} Int. Workshop on the Numerical Solution of Markov Chains, (Raleigh, NC, January 1995).

    This paper describes a procedure for the compositional derivation of an aggregated Markov process underlying a PEPA model that avoids the generation of the whole state space.

17. J. HILLSTON and V. MERTSIOTAKIS, "A Simple Time Scale Decomposition Technique for Stochastic Process Algebras," The Computer Journal, **38**(7), 1995. Special Issue: Proc. 3^{rd} Process Algebra and Performance Modeling Workshop, (Edinburgh, June 1995).

18. C.A.R. HOARE, "Communicating Sequential Processes," Prentice-Hall, (1985).

19. C-C. JOU and S.A. SMOLKA, "Equivalences, Congruences and Complete Axiomatizations of Probabilistic Processes," CONCUR'90, vol. 458 of LNCS, pp. 367–383. Springer-Verlag, (August 1990).

20. R. MILNER, "Communication and Concurrency," Prentice Hall, 1989.

This book describes CCS and the main features of the language. It is a good starting point for anyone wishing to learn about process algebra.

21. X. NICOLLIN and J. SIFAKIS, "An Overview and Synthesis on Timed Process Algebra," Real-Time: Theory in Practice, vol. 600 of LNCS, Springer Verlag, (1991).

This paper compares different timed extensions of process algebra. These extensions differ from SPAs since time progression and action execution are orthogonal concepts.

22. B. PLATEAU, J-M. FOURNEAU, and K-H. LEE, "PEPS: A Package for Solving Complex Markov Models of Parallel Systems," Proc. 4th Int. Conf. on Modeling Techniques and Tools for Computer Performance Evaluation, pp. 341–360, (1988).

23. M. RETTELBACH, "Probabilistic Branching in Markovian Process Algebras," The Computer Journal, **38**(7), 1995. Special Issue: Proc. 3rd Process Algebra and Performance Modeling Workshop, (Edinburgh, June 1995).

24. M.L. RETTELBACH and M. SIEGLE, "Compositional Minimal Semantics for the Stochastic Process Algebra TIPP," Proc. 2nd Process Algebra and Performance Modeling Workshop, (Erlangen, July 1994).

25. M. SERENO, "Towards a Product Form Solution for Stochastic Process Algebras," The Computer Journal, **38**(7), 1995. Special Issue: Proc. 3rd Process Algebra and Performance Modeling Workshop, (Edinburgh, June 1995).