15 Tackling state space explosion in PEPA models

In this lecture note we consider different approaches that are used to try to overcome the state space explosion problem in Markovian based models, using PEPA as an example.

Like all discrete state based modelling techniques, stochastic process algebra models are subject to the problem of state space explosion — the generated models may be intractable because of their size. A variety of techniques have been proposed for tackling this problem in the context of stochastic process algebra. Below we briefly discuss three of them:

- state space reduction via aggregation
- stochastic simulation over the state space
- fluid approximation of the state space.

15.1 State space reduction via aggregation

The state space explosion problem arises because although the compositionality of SPA can greatly aid model construction, in general the compositionality does not assist in the model solution and the resulting models may be too large to solve. This has led to research into how model simplification and aggregation techniques can be applied in the process algebra setting. Many such techniques are known in the context of Markov processes but are based on conditions expressed in terms of the generator matrix. This can mean that you have to construct the whole matrix before you can recognise that it has a property that allows you to decompose it. Moreover application of these techniques often relies on the expertise of the modeller. The challenge for SPA has been to define such model manipulation techniques in the context of the process algebra, in such a way that it can subsequently be applied automatically. Some significant results have been achieved in this area through the use of equivalence relations which provide the basis for comparing and manipulating models within a formal framework. Here we just consider a simple approach to model aggregation.

15.1.1 Model aggregation

The basic idea of model aggregation is that you partition the state space of the model, grouping together states that are in some way similar to each other. Formally this can be expressed in terms of an equivalence relation, so that you are grouping together states that have equivalent behaviour. Each partition is called a *macro-state*, since it represents one or more of the original states in the state space. The *aggregated process* is then a new stochastic process which operates only over the macro-states.

This approach has much intuitive appeal since it is conceptually simple and can lead to dramatic reduction in the size of the state. However, in general the new stochastic process that is created may bear little relationship to the original, and in particular, *may no longer be a Markov process.* In order to avoid this problem the partition has to satisfy a condition called *lumpability.* The technical details of lumpability are beyond the scope of this course, but you need to be aware that without this condition aggregation may provide a very poor approximation.

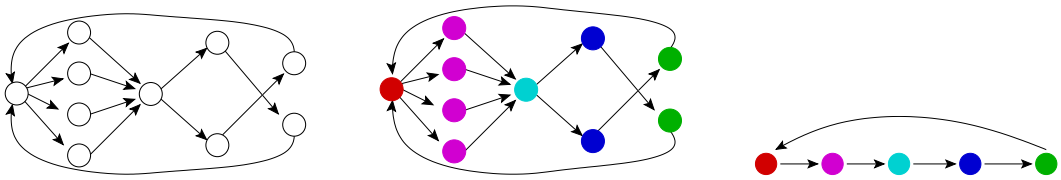

Figure 31: Schematic view of aggregation

15.1.2 State representation

The basis of aggregation is the observation that it can be sufficient to consider the behaviour of one element within an equivalence class of elements who all behave in the same way. The simplest way in which such equivalence classes arise is if we have repeated instances of identical components within the model. For this case, for PEPA models we have developed an automatic method which generates the Markov process corresponding to the equivalence classes, rather than the individual states, on-the-fly.

In order to apply this approach we must switch our state representation from being the syntactic terms of the PEPA process to an alternative state representation. In the new representation we simply count the number of components which are in each of their local states or derivatives. If there is only one instance of each type of component then this does not gain us anything, but when there are multiple instances the gain can be considerable.

In the *numerical vector form* there is one entry for each local derivative of each type of component in the model. The entries in the vector are no longer syntactic terms representing the local derivative of the sequential component, but the *number* of components currently exhibiting this local derivative. This is essentially the same as the *marking* used to capture the state of a SPN model. Because of the syntactic nature of SPA models the identity of components is captured by default by their position in the syntactic expression. In this state representation we instead treat the components as identity-less, just like tokens in an SPN.

To illustrate this new state representation consider the small example defined below, consisting of interacting processors and resources. For simplicity here we assume that both the processors and the resources have the same apparent rate for $task1$

$$\begin{aligned} Processor_0 &\stackrel{def}{=} (task1, r_1).Processor_1 \\ Processor_1 &\stackrel{def}{=} (task2, r_2).Processor_0 \\ Resource_0 &\stackrel{def}{=} (task1, r_1).Resource_1 \\ Resource_1 &\stackrel{def}{=} (reset, s).Resource_0 \end{aligned}$$

$$(Resource_0 \parallel Resource_0) \underset{\{task1\}}{\bowtie} (Processor_0 \parallel Processor_0)$$

In the numerical vector form, shown in Figure 32, the initial state is $(2, 0, 2, 0)$ where the entries in the vector are counting the number of $Resource_0$, $Resource_1$, $Processor_0$, $Processor_1$ local derivatives respectively, exhibited in the current state. If we consider the

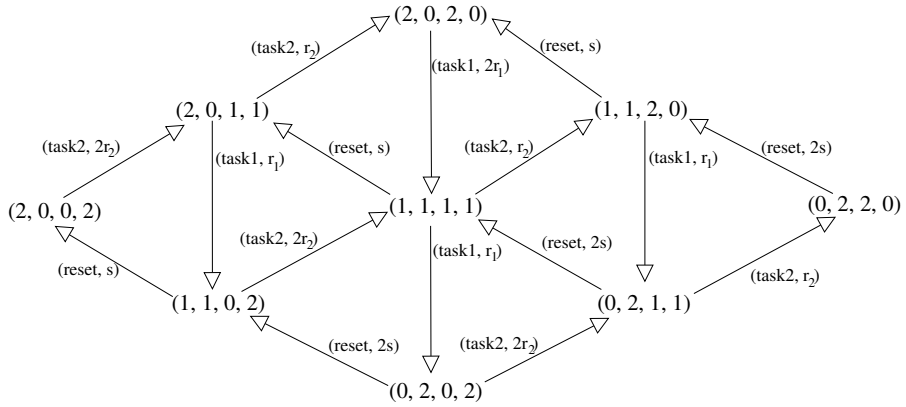

Figure 32: Numerical vector form state representation

state $(1, 1, 1, 1)$ it is representing four distinct syntactic states

$$(Resource_0, Resource_1, Processor_0, Processor_1)$$
$$(Resource_1, Resource_0, Processor_0, Processor_1)$$
$$(Resource_0, Resource_1, Processor_1, Processor_0)$$
$$(Resource_1, Resource_0, Processor_1, Processor_0)$$

It is clear here to see that all the four syntactic states will have exactly the same behaviour in terms of which activities they are able to do and the rates at which they will do them. In this case, it follows very straightforwardly that the condition of lumpability is satisfied and the stochastic process based on the numerical vector form is a Markov process. The lumpability condition also guarantees that the solution of the aggregated process (i.e. that based on the numerical vector form) is *exact*, in the sense that the steady state probability that it calculates for being in a macro-state is exactly the same as the steady state probability of the corresponding original states in the original Markov process.

15.2 Stochastic simulation over the state space

The aggregated Markov process underlying a PEPA model can be much more compact than the direct representation of the derivation graph in the naive Markov process. Nevertheless it can still rapidly exceed the capabilities of current numerical solution tools, especially in circumstances when there are large numbers of instances of components within the model. In this situation it may be possible to switch to alternative means of analysis to derive performance measures. For example, stochastic simulation may be used to study the system, rather than numerical solution to find the probability distribution. As we saw in a previous lecture note, this has the disadvantage that each run of the simulation generates only a single trajectory within the state space, rather than the consideration of all possible behaviours which is encompassed in the numerical solution. Thus simulation necessitates multiple runs of the model in order to derive statistically significant results. However in situations when numerical solution becomes infeasible because the state space is too large, simulation's ability to avoid explicit representation of the entire state space is invaluable.

When we use stochastic simulation as a means to solve large PEPA models we can take advantage of the fact that the stochastic process underlying the model is a Markov process. The memoryless property of the Markov process means that we do not need to maintain an *event list*. The definition of a Markov process and the memoryless property means that in order to know what might happen next we only need to know the current state and the possible activities in that state. In this case the simulation algorithm is particularly simple and relatively efficient. In the PEPA Plug-in for Eclipse tool the stochastic simulation is based on the numerical vector form rather than the syntactic states.

To see how the simulation algorithm works consider a PEPA model which enables a number of possible activities $(\alpha_1, r_1), (\alpha_2, r_2), \ldots, (\alpha_n, r_n)$. Using the *superposition principle* of exponential distributions, we know that the time until *something* happens is the minimum of the exponential distributions governing each of the activities, which is itself an exponential distribution with rate $r_1 + r_2 + \cdots + r_n$. Thus we make one sample from the random number distribution and use it to create a random variate sample for the exponential distribution with parameter $r_1 + r_2 + \cdots + r_n$. Simulation time is advanced to the current time plus the value of this sample.

Next we draw a second random number sample and use it to select which of the activities has been completed. Let $R = r_1 + r_2 + \cdots + r_n$ and let ρ be the sample from the random number generator. If $\rho \leq \frac{r_1}{R}$ then perform activity α_1; if $\frac{r_1}{R} < \rho \leq \frac{r_1+r_2}{R}$, perform activity α_2; etc. Performing the activity will lead to a new state of the PEPA model and the algorithm can be applied again considering the activities that are enabled in that state, $(\beta_1, s_1), (\beta_2, s_2), \ldots, (\beta_m, s_m)$, etc.

Thus we need only draw two random numbers for each step of the simulation algorithm:

- the first determines the delay until the next activity completes,
- the second determines which activity that will be.

Stochastic simulation is one of the options on offer in the PEPA Plug-in for Eclipse under the **Scalable analysis** option, or if you want to produce graphs you should use the **Time Series Wizard** to help you. Figure 33 shows the results of a simulation analysis of the simple web service example from Lecture Note 11, with many more copies of both the application and the web service.

15.3 Fluid Approximation

Recent work on PEPA has developed another form of approximation for PEPA models based on the internal representation in numerical vector form, which is particularly suitable for systems which are comprised of multiple instances of components. This *fluid approximation* is capable of handling models which are many orders of magnitude larger than what can be solved numerically and is significantly more efficient than stochastic simulation. The crucial step in the approximation is to treat the state variable, the numerical vector form, as being subject to continuous rather than discrete change. Once this step is taken the evolution of the system through the state space can be represented by a set of ordinary differential equations (ODEs).

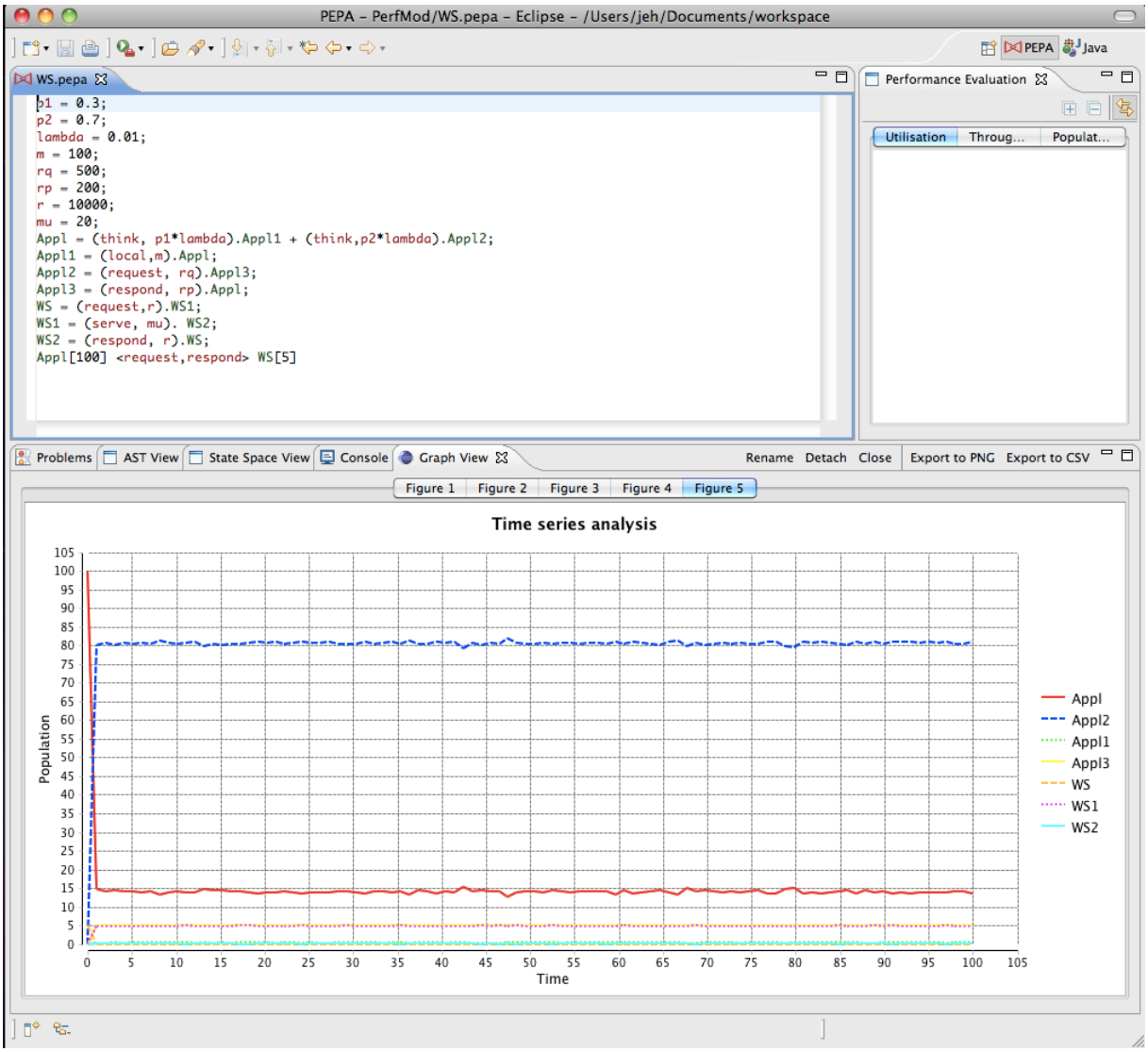

Figure 33: The results of a simulation analysis of the simple web service example from Lecture Note 11, with many more copies of both the application and the web service.

This shift from the discrete, stochastic representation of a Markov process to the continuous, deterministic representation of a set of ODEs may seem surprising at first. The use of continuous variables is clearly an approximation since a variable of the system, for example the number of idle servers, or the number of occupied threads, will always be a natural number in reality. The events which impact on a variable such as the number of idle servers (e.g. the arrival of a customer) cause a discrete change in the system. In fact the effect of an activity is at most to increment or decrement elements of the numerical vector in steps of one. Nevertheless, when there are large numbers of instances of entities in the system, these changes are relatively small and we can approximate the behaviour by considering the movement between states to be continuous, rather than occurring in discontinuous jumps. Moreover when there are many, many components in the system the events will be occurring very frequently, further justifying the continuous approximation and the relative impact of the events mean that treating these changes as continuous is justified. In PEPA models with many instances of components the variability within the system grows less and the stochastic system tends to behaviour more and more like its average, or expected, behaviour. This is exactly the behaviour that is captured by the set of ODEs.

The numerical vector form of state representation is an intermediate step to achieving the fluid approximation. Considering these states of the process and the activities which are enabled, and the states they lead to, we are able to construct an *activity matrix* which records the impact of each activity type on the number of each component type. From this the appropriate system of ODEs is derived.

The sets of ODEs generated are rarely amenable to analytical solution but they are readily solved using numerical integration. Such analysis produces a time series of values for each of the continuous variables. Related to the conventional performance analyses conducted based on numerical solution of Markov processes these values correspond to the number of instances of each of the local states of the components of the model, which can be regarded as a form of utilisation. However, we have also been able to develop rigorous methods to derive more sophisticated performance indices such as throughput and average response time, from the numerical integration of the ODEs, although these will not be discussed here.

15.3.1 Small example revisited

Let us consider again the small example considered earlier, assuming now that there are large numbers of processors and resources:

$$\begin{aligned}
Processor_0 &\stackrel{def}{=} (task1, r_1).Processor_1 \\
Processor_1 &\stackrel{def}{=} (task2, r_2).Processor_0 \\
Resource_0 &\stackrel{def}{=} (task1, r_1).Resource_1 \\
Resource_1 &\stackrel{def}{=} (reset, s).Resource_0
\end{aligned}$$

$$Processor_0[N_P] \underset{\{task1\}}{\bowtie} Resource_0[N_R]$$

Let x_1 denote the number of $Processor_0$ entities, x_2 the number of $Processor_1$ entities,

	$task_1$	$task_2$	$reset$	
$Processor_0$	-1	$+1$	0	x_1
$Processor_1$	$+1$	-1	0	x_2
$Resource_0$	-1	0	$+1$	x_3
$Resource_1$	$+1$	0	-1	x_4

Figure 34: Activity matrix for the simple Processor-Resource model

x_3 the number of Res_0 entities and x_4 the number of $Resource_1$ entities. The activity matrix corresponding the component definitions is shown in Figure 34.

The activity matrix has a row for each local state and a column for each action type in the model. The entry in the (i, j)-th position in the matrix can be $-1, 0$, or 1.

- If the entry is -1 it means that this local state undertakes an activity of that type and so when the activity is completed there will be one less instance of this local state.
- If the entry is 0 this local state is not involved in this activity.
- If the entry is 1 it means that this local state is produced when the activity of that type is completed, so there will be one more instance of this local state.

From the matrix, we derive each differential equation in turn. For state variable x_i, consider row i. Each non-zero entry in the row will result in one term within the equation. The negative entries in the column will show which other local states are involved in this activity.

$$\begin{aligned}
\frac{dx_1(t)}{dt} &= -r_1 \min(x_1(t), x_3(t)) + r_2 x_2(t) \\
\frac{dx_2(t)}{dt} &= r_1 \min(x_1(t), x_3(t)) - r_2 x_2(t) \\
\frac{dx_3(t)}{dt} &= -r_1 \min(x_1(t), x_3(t)) + s x_4(t) \\
\frac{dx_4(t)}{dt} &= x_1 \min(x_1(t), x_3(t)) - s x_4(t)
\end{aligned}$$

Note that the form of the system of ODEs is independent of the number of components included in the initial configuration of the model. The only impact of changing the number of instances of each component type is to alter the initial conditions. Thus, if there are initially 1024 processors, all starting in state $Processor_0$ and 512 resources, all of which start in state $Resource_0$, the initial conditions will be:

$$x_1(0) = 1024 \quad x_2(0) = 0 \quad x_3(0) = 512 \quad x_4(0) = 0$$

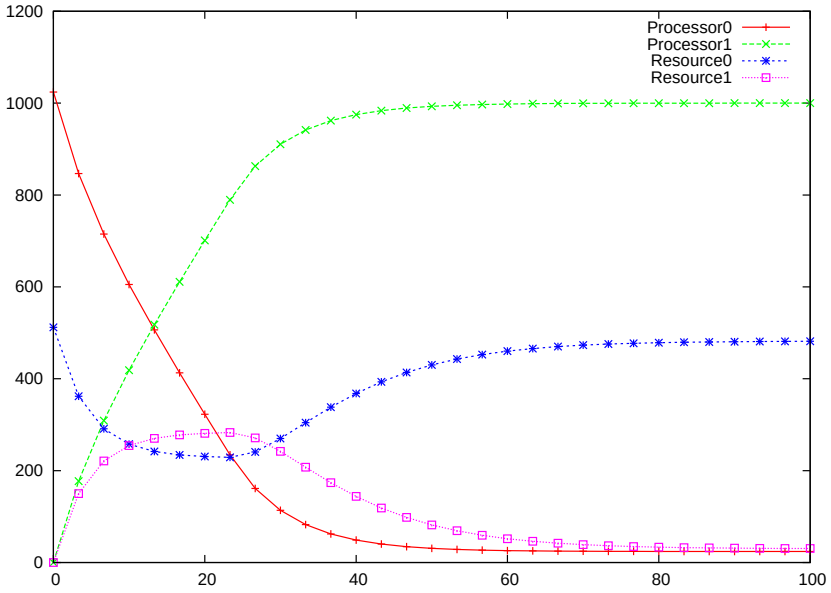

Figure 35: Graph showing the changing numbers of copies of $Processor_0$, $Processor_1$, $Resource_0$ and $Resource_1$ as a function of time, obtained by numerically integrating the differential equations for this system. The values of the rates were $r_1 = 0.125$, $r_2 = 0.003$ and $s = 0.1$.

Numerically integrating the differential equations for this system to generate a time series plot for the first 100 seconds of the system evolution starting from the above initial value problem produces the graph shown in Figure 35.

Jane Hillston ⟨Jane.Hillston@ed.ac.uk⟩. November 12, 2012.