Performance Modelling — Lecture 15: Tackling state space explosion in PEPA models

Jane Hillston
School of Informatics
The University of Edinburgh
Scotland

13th March 2017

State Space Explosion

The numerical solution of CTMC models such as those built using stochastic Petri nets and stochastic process algebras, like PEPA, relies on construction of the $N \times N$ infinitesimal generator matrix $\mathbf{Q}$, and the N-dimensional probability vector π, where N is the size of the state space.

State Space Explosion

The numerical solution of CTMC models such as those built using stochastic Petri nets and stochastic process algebras, like PEPA, relies on construction of the $N \times N$ infinitesimal generator matrix $\mathbf{Q}$, and the N-dimensional probability vector π, where N is the size of the state space.

Unfortunately, the size of these entities often exceeds what can be handled in memory.

State Space Explosion

The numerical solution of CTMC models such as those built using stochastic Petri nets and stochastic process algebras, like PEPA, relies on construction of the $N \times N$ infinitesimal generator matrix $\mathbf{Q}$, and the N-dimensional probability vector π, where N is the size of the state space.

Unfortunately, the size of these entities often exceeds what can be handled in memory.

This problem is known as state space explosion.

(All discrete state modelling approaches are prone to this problem.)

A simple example: processors and resources

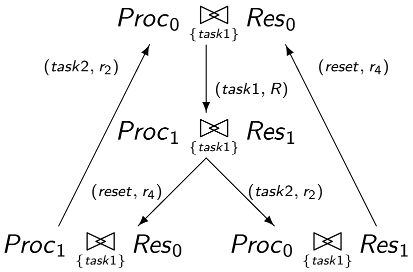
$$
\begin{aligned}
Proc_0 &\stackrel{def}{=} (task1, r_1).Proc_1 \\
Proc_1 &\stackrel{def}{=} (task2, r_2).Proc_0 \\
Res_0 &\stackrel{def}{=} (task1, r_3).Res_1 \\
Res_1 &\stackrel{def}{=} (reset, r_4).Res_0
\end{aligned}
$$

$$
Proc_0 \underset{\{task1\}}{\bowtie} Res_0
$$

A simple example: processors and resources

$$
\begin{aligned}
Proc_0 &\stackrel{def}{=} (task1, r_1).Proc_1 \\
Proc_1 &\stackrel{def}{=} (task2, r_2).Proc_0 \\
Res_0 &\stackrel{def}{=} (task1, r_3).Res_1 \\
Res_1 &\stackrel{def}{=} (reset, r_4).Res_0
\end{aligned}
$$

$$Proc_0 \underset{\{task1\}}{\bowtie} Res_0$$

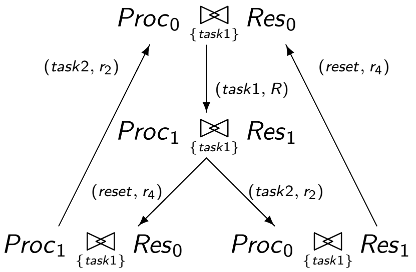


$$R = \min(r_1, r_3)$$

A simple example: processors and resources

$$
\begin{aligned}
Proc_0 &\stackrel{def}{=} (task1, r_1).Proc_1 \\
Proc_1 &\stackrel{def}{=} (task2, r_2).Proc_0 \\
Res_0 &\stackrel{def}{=} (task1, r_3).Res_1 \\
Res_1 &\stackrel{def}{=} (reset, r_4).Res_0
\end{aligned}
$$

$$Proc_0 \underset{\{task1\}}{\bowtie} Res_0$$



$$R = \min(r_1, r_3)$$

$$
\mathbf{Q} = \begin{pmatrix}
-R & R & 0 & 0 \\
0 & -(r_2 + r_4) & r_4 & r_2 \\
r_2 & 0 & -r_2 & 0 \\
r_4 & 0 & 0 & -r_4
\end{pmatrix}
$$

Simple example : multiple instances

$$\begin{aligned} Proc_0 &\stackrel{def}{=} (task1, r_1).Proc_1 \\ Proc_1 &\stackrel{def}{=} (task2, r_2).Proc_0 \\ Res_0 &\stackrel{def}{=} (task1, r_3).Res_1 \\ Res_1 &\stackrel{def}{=} (reset, r_4).Res_0 \end{aligned}$$

$$Proc_0[N_P] \underset{\{task1\}}{\bowtie} Res_0[N_R]$$

Simple example : multiple instances

$$
\begin{aligned}
Proc_0 &\stackrel{def}{=} (task1, r_1).Proc_1 \\
Proc_1 &\stackrel{def}{=} (task2, r_2).Proc_0 \\
Res_0 &\stackrel{def}{=} (task1, r_3).Res_1 \\
Res_1 &\stackrel{def}{=} (reset, r_4).Res_0
\end{aligned}
$$

$$Proc_0[N_P] \underset{\{task1\}}{\bowtie} Res_0[N_R]$$

CTMC interpretation

Processors (N_P)	Resources (N_R)	States ($2^{N_P+N_R}$)
1	1	4
2	1	8
2	2	16
3	2	32
3	3	64
4	3	128
4	4	256
5	4	512
5	5	1024
6	5	2048
6	6	4096
7	6	8192
7	7	16384
8	7	32768
8	8	65536
9	8	131072
9	9	262144
10	9	524288
10	10	1048576

Simple example : multiple instances

$$Proc_0 \stackrel{def}{=} (task1, r_1).Proc_1$$
$$Proc_1 \stackrel{def}{=} (task2, r_2).Proc_0$$
$$Res_0 \stackrel{def}{=} (task1, r_3).Res_1$$
$$Res_1 \stackrel{def}{=} (reset, r_4).Res_0$$

$$Proc_0[N_P] \underset{\{task1\}}{\bowtie} Res_0[N_R]$$

CTMC interpretation

Processors (N_P)	Resources (N_R)	States ($2^{N_P+N_R}$)
1	1	4
2	1	8
2	2	16
3	2	32
3	3	64
4	3	128
4	4	256
5	4	512
5	5	1024
6	5	2048
6	6	4096
7	6	8192
7	7	16384
8	7	32768
8	8	65536
9	8	131072
9	9	262144
10	9	524288
10	10	1048576

The size of state space: $2^{N_P} \times 2^{N_R}$.

Tackling state space explosion

- To overcome state-space explosion problem in CTMCs, many mathematical tools and approaches have been proposed.

Tackling state space explosion

- To overcome state-space explosion problem in CTMCs, many mathematical tools and approaches have been proposed.
- We will use the stochastic process algebra, PEPA as an example, and give an overview of three different approaches to tackling the state space explosion problem.

Tackling state space explosion

- To overcome state-space explosion problem in CTMCs, many mathematical tools and approaches have been proposed.
- We will use the stochastic process algebra, PEPA as an example, and give an overview of three different approaches to tackling the state space explosion problem.
  - state space reduction via aggregation;

Tackling state space explosion

- To overcome state-space explosion problem in CTMCs, many mathematical tools and approaches have been proposed.
- We will use the stochastic process algebra, PEPA as an example, and give an overview of three different approaches to tackling the state space explosion problem.
  - state space reduction via aggregation;
  - stochastic simulation over the discrete state space;

Tackling state space explosion

- To overcome state-space explosion problem in CTMCs, many mathematical tools and approaches have been proposed.
- We will use the stochastic process algebra, PEPA as an example, and give an overview of three different approaches to tackling the state space explosion problem.
  - state space reduction via aggregation;
  - stochastic simulation over the discrete state space;
  - fluid approximation of the state space.

Aggregation and lumpability

- Model aggregation: partition the state space of a model, and replace each set of states by one macro-state

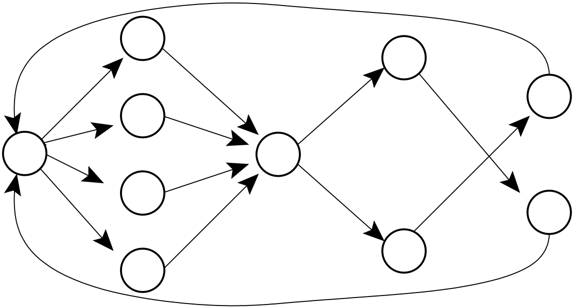
Aggregation and lumpability

- Model aggregation: partition the state space of a model, and replace each set of states by one macro-state
- This is not as straightforward as it may seem if we wish the aggregated process to still be a Markov process — an arbitrary partition will not in general preserve the Markov property.

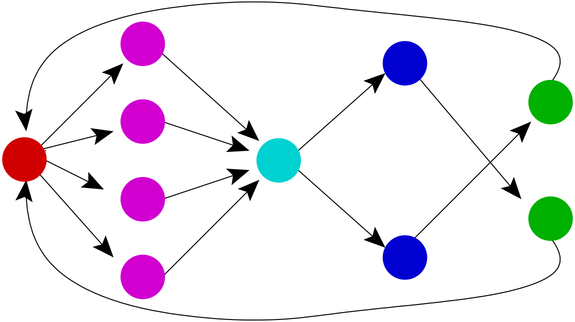
Aggregation and lumpability

- Model aggregation: partition the state space of a model, and replace each set of states by one macro-state
- This is not as straightforward as it may seem if we wish the aggregated process to still be a Markov process — an arbitrary partition will not in general preserve the Markov property.
- In order to preserve the Markov property we must ensure that the partition satisfies a condition called lumpability.

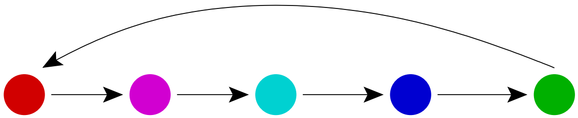
Reducing by lumpability



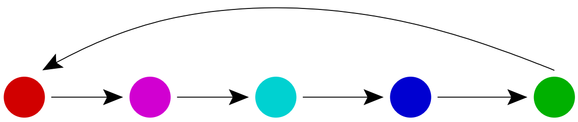
Reducing by lumpability



Reducing by lumpability



Reducing by lumpability



Arbitrarily lumping the states of a Markov chain, will typically give rise to a stochastic process which no longer satisfies the Markov condition.

Losing identity

The syntactic nature of PEPA makes models easily understood by humans, but not so convenient for approaches such as aggregation and simulation.

Losing identity

The syntactic nature of PEPA makes models easily understood by humans, but not so convenient for approaches such as aggregation and simulation.

In particular when we have many instances of the same component type, in the PEPA expression these instances are distinguished by their location (position from left to right) in the expression.

Losing identity

The syntactic nature of PEPA makes models easily understood by humans, but not so convenient for approaches such as aggregation and simulation.

In particular when we have many instances of the same component type, in the PEPA expression these instances are distinguished by their location (position from left to right) in the expression.

However, in general we do not care which such instance is involved in an event, just that one of them is, i.e. it is sufficient to count the instances that are in the possible local states.

Losing identity

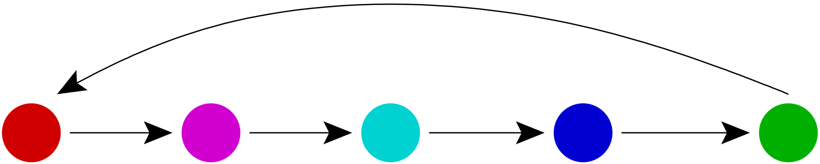
The syntactic nature of PEPA makes models easily understood by humans, but not so convenient for approaches such as aggregation and simulation.

In particular when we have many instances of the same component type, in the PEPA expression these instances are distinguished by their location (position from left to right) in the expression.

However, in general we do not care which such instance is involved in an event, just that one of them is, i.e. it is sufficient to count the instances that are in the possible local states.

Thus we change to a state representation which is a numerical state vector, analogous to the marking in a SPN.

Reducing by lumpability



When we use the numerical vector state representation for PEPA we group together those expressions that have the same counts for each of the local states and we are certain that the partition that we induce on the state space is lumpable and so the lumped process is still a Markov process.

Example revisited

$$
\begin{aligned}
Proc_0 &\stackrel{def}{=} (task1, r_1).Proc_1 \\
Proc_1 &\stackrel{def}{=} (task2, r_2).Proc_0 \\
Res_0 &\stackrel{def}{=} (task1, r_1).Res_1 \\
Res_1 &\stackrel{def}{=} (reset, s).Res_0
\end{aligned}
$$

$$
(Res_0 \parallel Res_0) \underset{\{task1\}}{\bowtie} (Proc_0 \parallel Proc_0)
$$

Numerical vector form

For our example model:

$$\mathbf{m} = (\mathbf{m}[Proc_0], \mathbf{m}[Proc_1], \mathbf{m}[Res_0], \mathbf{m}[Res_1])\,.$$

Numerical vector form

For our example model:

$$\mathbf{m} = (\mathbf{m}[Proc_0], \mathbf{m}[Proc_1], \mathbf{m}[Res_0], \mathbf{m}[Res_1]) \,.$$

When $N_P = N_R = 2$, the system equation of the model determines the starting state:

$$\mathbf{m} = (N_P, 0, N_R, 0) = (2, 0, 2, 0)$$

We can apply the possible activities in each of the states until we find all possible states.

$$\begin{array}{lll} \mathbf{s}_1 = (2,0,2,0), & \mathbf{s}_2 = (1,1,1,1), & \mathbf{s}_3 = (1,1,2,0), \\ \mathbf{s}_4 = (1,1,0,2), & \mathbf{s}_5 = (0,2,1,1), & \mathbf{s}_6 = (2,0,1,1), \\ \mathbf{s}_7 = (0,2,0,2), & \mathbf{s}_8 = (0,2,2,0), & \mathbf{s}_9 = (2,0,0,2). \end{array}$$

Numerical vector form

The initial state is $(2, 0, 2, 0)$ where the entries in the vector are counting the number of Res_0, Res_1, $Proc_0$, $Proc_1$ local derivatives respectively, exhibited in the current state.

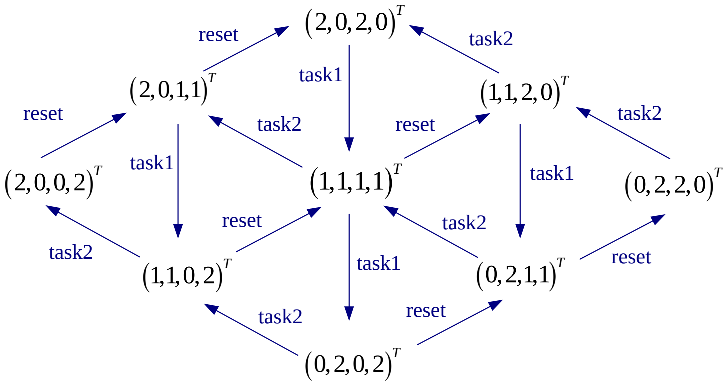
Numerical vector form

The initial state is $(2, 0, 2, 0)$ where the entries in the vector are counting the number of Res_0, Res_1, $Proc_0$, $Proc_1$ local derivatives respectively, exhibited in the current state.

If we consider the state $(1, 1, 1, 1)$ it is representing four distinct syntactic states

$$(Res_0, Res_1, Proc_0, Proc_1)$$
$$(Res_1, Res_0, Proc_0, Proc_1)$$
$$(Res_0, Res_1, Proc_1, Proc_0)$$
$$(Res_1, Res_0, Proc_1, Proc_0)$$

The resulting state space



The size of the state space: $(N_P + d_P - 1)^{d_P - 1} \times (N_R + d_R - 1)^{d_R - 1}$.

Solution of an aggregated model

Once we have the state space of the aggregated model we construct the CTMC in the obvious way — associating one state with each node in the aggregated state transition diagram.

Solution of an aggregated model

Once we have the state space of the aggregated model we construct the CTMC in the obvious way — associating one state with each node in the aggregated state transition diagram.

This CTMC will typically have a smaller state space than the one derived from the original state representation as a derivative graph, and certainly no larger.

Solution of an aggregated model

Once we have the state space of the aggregated model we construct the CTMC in the obvious way — associating one state with each node in the aggregated state transition diagram.

This CTMC will typically have a smaller state space than the one derived from the original state representation as a derivative graph, and certainly no larger.

The steady state probability distribution can then be derived in the usual way by solving the global balance equations.

Solution of an aggregated model

Once we have the state space of the aggregated model we construct the CTMC in the obvious way — associating one state with each node in the aggregated state transition diagram.

This CTMC will typically have a smaller state space than the one derived from the original state representation as a derivative graph, and certainly no larger.

The steady state probability distribution can then be derived in the usual way by solving the global balance equations.

The solution gives you the probability of being in the set of states that have the same behaviour.

Simulation in PEPA

When we simulate PEPA models we are simulating the underlying Markov process, avoiding the construction of the whole state space at once, instead finding the states step-by-step as the simulation progresses.

Simulation in PEPA

When we simulate PEPA models we are simulating the underlying Markov process, avoiding the construction of the whole state space at once, instead finding the states step-by-step as the simulation progresses.

Because we are working in the Markovian context we can take advantage of the memoryless property.

Simulation in PEPA

When we simulate PEPA models we are simulating the underlying Markov process, avoiding the construction of the whole state space at once, instead finding the states step-by-step as the simulation progresses.

Because we are working in the Markovian context we can take advantage of the memoryless property.

This means that we do not need to maintain an event list.

Simulation in PEPA

When we simulate PEPA models we are simulating the underlying Markov process, avoiding the construction of the whole state space at once, instead finding the states step-by-step as the simulation progresses.

Because we are working in the Markovian context we can take advantage of the memoryless property.

This means that we do not need to maintain an event list.

In this case the simulation algorithm is particularly simple and relatively efficient.

The Gillespie Stochastic Simulation Algorithm

You can think of the simulation of PEPA as being a process-based simulation.

The Gillespie Stochastic Simulation Algorithm

You can think of the simulation of PEPA as being a process-based simulation.

Instead of an event list the simulation engine keeps the state of the system and so knows for each component what activity or activities it currently enables (for shared activities it will check that all participating components are able to undertake the actions).

The Gillespie Stochastic Simulation Algorithm

You can think of the simulation of PEPA as being a process-based simulation.

Instead of an event list the simulation engine keeps the state of the system and so knows for each component what activity or activities it currently enables (for shared activities it will check that all participating components are able to undertake the actions).

From this list of possible activities it will select one to execute according to the race policy and then update the state accordingly, modifying the list of current activities as necessary.

Two Observations

If we have a number of possible activities $(\alpha_1, r_1), (\alpha_2, r_2), \ldots, (\alpha_n, r_n)$ enabled in the current state, then we know from the superposition principle for the exponential distribution that the time until something happens is governed by an exponential distribution with rate $r_1 + r_2 + \cdots + r_n$.

Two Observations

If we have a number of possible activities $(\alpha_1, r_1), (\alpha_2, r_2), \ldots, (\alpha_n, r_n)$ enabled in the current state, then we know from the superposition principle for the exponential distribution that the time until something happens is governed by an exponential distribution with rate $r_1 + r_2 + \cdots + r_n$.

We also know that the probability that it is the activity of type α_i is

$$\frac{r_i}{r_1 + r_2 + \cdots + r_n}.$$

The Gillespie Stochastic Simulation Algorithm

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

The Gillespie Stochastic Simulation Algorithm

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 Gillespie Stochastic Simulation Algorithm

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.

Fluid Approximation

The third approach to tackling state space explosion that we consider is the use of fluid or continuous approximation.

Fluid Approximation

The third approach to tackling state space explosion that we consider is the use of fluid or continuous approximation.

Here the key idea is to approximate the behaviour of a discrete event system which jumps between discrete states by a continuous system which moves smoothly over a continuous state space.

Continuously varying counting variables

When this is applied in performance models the state space is usually characterised by counting variables:

- the number of customers in a queue,
- the number of servers who are busy, or
- the number of local derivatives in a particular state in a PEPA model.

Continuously varying counting variables

When this is applied in performance models the state space is usually characterised by counting variables:

- the number of customers in a queue,
- the number of servers who are busy, or
- the number of local derivatives in a particular state in a PEPA model.

Allowing continuous variables for these quantities might seem odd to begin with — what does it mean for 0.65 servers to be busy? — but when we think of it as the average it becomes easier to interpret.

Fluid Approximation

When we have multiple instances of components within a model, the effect of a state change by one component in the system becomes relatively small.

Fluid Approximation

When we have multiple instances of components within a model, the effect of a state change by one component in the system becomes relatively small.

Randomness in behaviour also begins to average out between the different components.

Fluid Approximation

When we have multiple instances of components within a model, the effect of a state change by one component in the system becomes relatively small.

Randomness in behaviour also begins to average out between the different components.

So we can use continuous state variables to approximate the discrete state space (assuming numerical state representation).

Fluid Approximation

When we have multiple instances of components within a model, the effect of a state change by one component in the system becomes relatively small.

Randomness in behaviour also begins to average out between the different components.

So we can use continuous state variables to approximate the discrete state space (assuming numerical state representation). We then use ordinary differential equations to represent the evolution of those variables over time.

Fluid approximation

- Use a more abstract state representation rather than the CTMC complete state space based on LTS: numerical vector form.

Fluid approximation

- Use a more abstract state representation rather than the CTMC complete state space based on LTS: numerical vector form.
- Assume that these state variables are subject to continuous rather than discrete change.

Fluid approximation

- Use a more abstract state representation rather than the CTMC complete state space based on LTS: numerical vector form.
- Assume that these state variables are subject to continuous rather than discrete change.
- No longer aim to calculate the probability distribution over the entire state space of the model.

Fluid approximation

- Use a more abstract state representation rather than the CTMC complete state space based on LTS: numerical vector form.
- Assume that these state variables are subject to continuous rather than discrete change.
- No longer aim to calculate the probability distribution over the entire state space of the model.

Appropriate for models in which there are large numbers of components of the same type.

Differential equations from PEPA models

- The PEPA definitions of the component specify the activities which can increase or decrease the number of components exhibited in the current state.
- The cooperations show when the number of instances of another component will have an influence on the evolution of this component.

Example revisited

$$
\begin{aligned}
Proc_0 &\stackrel{def}{=} (task1, r_1).Proc_1 \\
Proc_1 &\stackrel{def}{=} (task2, r_2).Proc_0 \\
Res_0 &\stackrel{def}{=} (task1, r_1).Res_1 \\
Res_1 &\stackrel{def}{=} (reset, r_4).Res_0
\end{aligned}
$$

$$
Proc_0[N_P] \underset{\{task1\}}{\bowtie} Res_0[N_R]
$$

Example revisited

$$
\begin{aligned}
Proc_0 &\stackrel{def}{=} (task1, r_1).Proc_1 \\
Proc_1 &\stackrel{def}{=} (task2, r_2).Proc_0 \\
Res_0 &\stackrel{def}{=} (task1, r_1).Res_1 \\
Res_1 &\stackrel{def}{=} (reset, r_4).Res_0
\end{aligned}
$$

$$
Proc_0[N_P] \underset{\{task1\}}{\bowtie} Res_0[N_R]
$$

- $task1$ decreases $Proc_0$ and Res_0
- $task1$ increases $Proc_1$ and Res_1
- $task2$ decreases $Proc_1$ and increases $Proc_0$
- $reset$ decreases Res_1 and increases Res_0

We can capture the relationship between activities and components in a matrix called the activity matrix which has one row for each component and one column for each activity.

Example revisited

$$Proc_0 \stackrel{def}{=} (task1, r_1).Proc_1$$
$$Proc_1 \stackrel{def}{=} (task2, r_2).Proc_0$$
$$Res_0 \stackrel{def}{=} (task1, r_1).Res_1$$
$$Res_1 \stackrel{def}{=} (reset, r_4).Res_0$$

$$Proc_0[N_P] \underset{\{task1\}}{\bowtie} Res_0[N_R]$$

ODE interpretation

$$\frac{dx_1}{dt} = -r_1 \min(x_1, x_3) + r_2\, x_2$$
$$x_1 = \text{no. of } Proc_1$$
$$\frac{dx_2}{dt} = r_1 \min(x_1, x_3) - r_2\, x_2$$
$$x_2 = \text{no. of } Proc_2$$
$$\frac{dx_3}{dt} = -r_1 \min(x_1, x_3) + r_4\, x_4$$
$$x_3 = \text{no. of } Res_0$$
$$\frac{dx_4}{dt} = r_1 \min(x_1, x_3) - r_4\, x_4$$
$$x_4 = \text{no. of } Res_1$$

We can capture the relationship between activities and components in a matrix called the activity matrix which has one row for each component and one column for each activity.

Activity matrix

Derivation of the system of ODEs representing the PEPA model can proceed via the activity matrix which records the influence of each activity on each component type/derivative.

The matrix has one row for each component type and one column for each activity type.

One ODE is generated corresponding to each row of the matrix, taking into account the negative entries in the non-zero columns as these are the components for which this is an exit activity.

Activity matrix for the small example

	$task_1$	$task_2$	$reset$	
$Proc_0$	-1	$+1$	0	x_1
$Proc_1$	$+1$	-1	0	x_2
Res_0	-1	0	$+1$	x_3
Res_1	$+1$	0	-1	x_4

Activity matrix to ODEs

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.

Activity matrix to ODEs

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.

Activity matrix to ODEs

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.

ODEs

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

- The form of ODEs is independent of the number of instances of components in the model.

ODEs

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

- The form of ODEs is independent of the number of instances of components in the model.
- The only impact of changing the number of instances is to alter the initial conditions.

Initialising the ODEs

Consider the model $Proc_0[100] \underset{\{task1\}}{\bowtie} Res_0[80]$.

There are initially 100 processors, all starting in state $Proc_0$ and 80 resources, all of which start in state Res_0.

Initialising the ODEs

Consider the model $Proc_0[100] \underset{\{task1\}}{\bowtie} Res_0[80]$.

There are initially 100 processors, all starting in state $Proc_0$ and 80 resources, all of which start in state Res_0.

Then we set the initial conditions of the ODEs to be:

$$x_1(0) = 100 \quad x_2(0) = 0 \quad x_3(0) = 80 \quad x_4(0) = 0$$

Initialising the ODEs

Consider the model $Proc_0[100] \underset{\{task1\}}{\bowtie} Res_0[80]$.

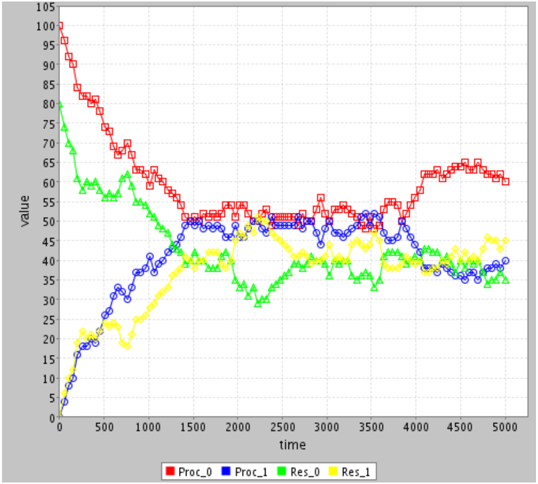
There are initially 100 processors, all starting in state $Proc_0$ and 80 resources, all of which start in state Res_0.

Then we set the initial conditions of the ODEs to be:

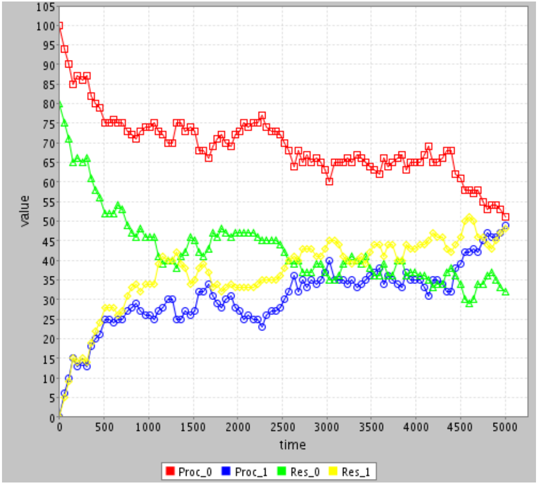
$$x_1(0) = 100 \quad x_2(0) = 0 \quad x_3(0) = 80 \quad x_4(0) = 0$$

The system of ODEs can then be given to any suitable numerical solver as an initial value problem.

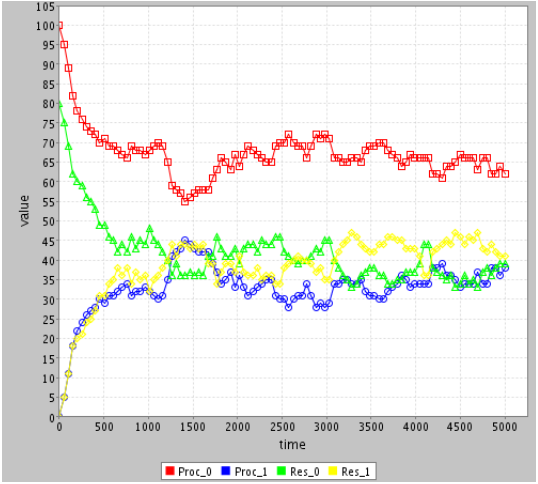
100 processors and 80 resources (simulation run A)



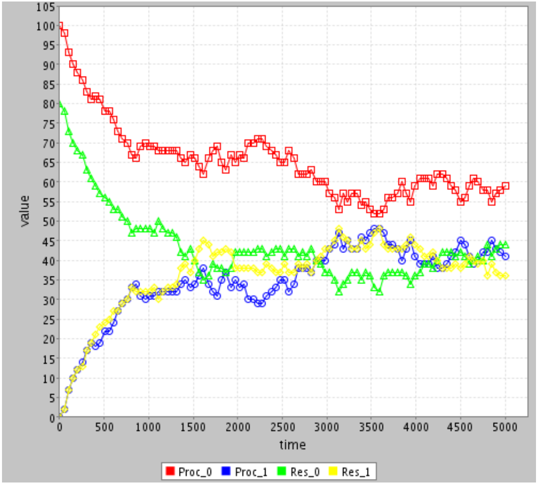
100 processors and 80 resources (simulation run B)



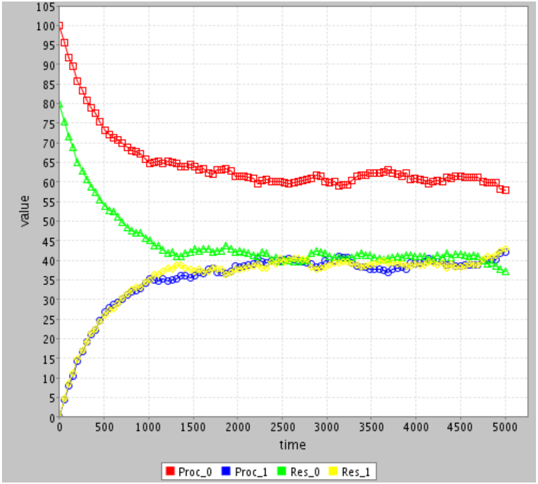
100 processors and 80 resources (simulation run C)



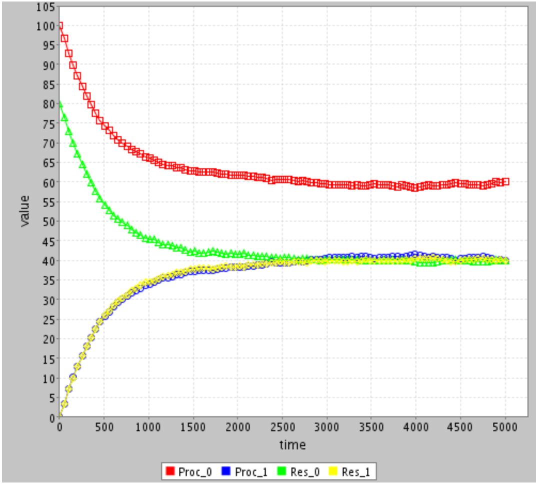
100 processors and 80 resources (simulation run D)



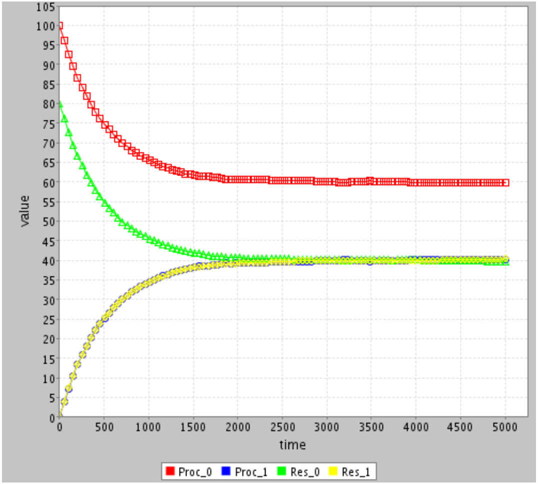
100 processors and 80 resources (average of 10 runs)



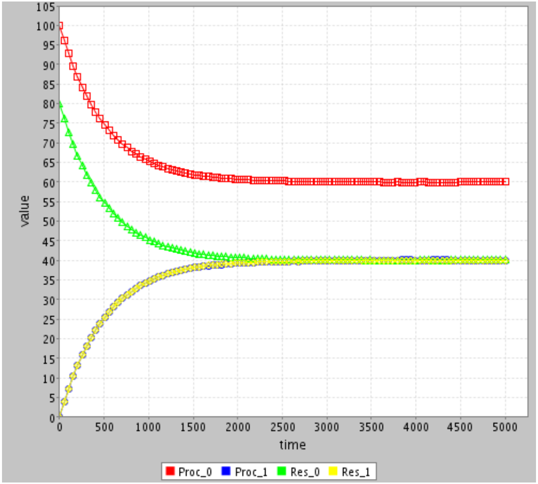
100 Processors and 80 resources (average of 100 runs)



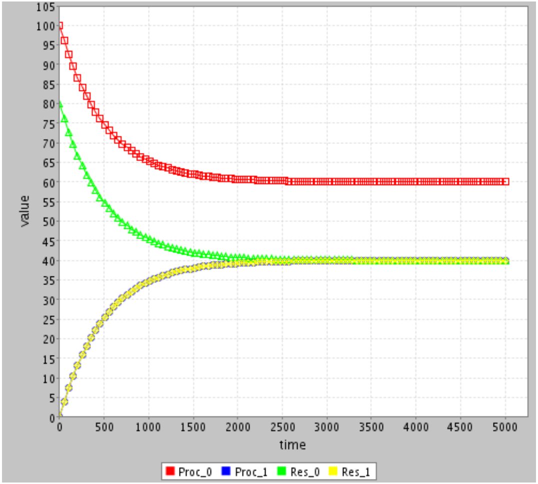
100 processors and 80 resources (average of 1000 runs)



100 processors and 80 resources (average of 10000 runs)



100 processors and 80 resources (ODE solution)



Deriving quantitative data

PEPA models can be analysed for quantified dynamic behaviour in a number of different ways.

Deriving quantitative data

PEPA models can be analysed for quantified dynamic behaviour in a number of different ways.

The language may be used to generate a Markov Process (CTMC).

Deriving quantitative data

PEPA models can be analysed for quantified dynamic behaviour in a number of different ways.

The language may be used to generate a Markov Process (CTMC).

PEPA MODEL —SOS rules→ LABELLED TRANSITION SYSTEM —state transition diagram→ CTMC **Q**

Deriving quantitative data

PEPA models can be analysed for quantified dynamic behaviour in a number of different ways.

The language also may be used to generate a stochastic simulation.

Deriving quantitative data

PEPA models can be analysed for quantified dynamic behaviour in a number of different ways.

The language also may be used to generate a stochastic simulation.

Deriving quantitative data

PEPA models can be analysed for quantified dynamic behaviour in a number of different ways.

The language may be used to generate a system of ordinary differential equations (ODEs).

Deriving quantitative data

PEPA models can be analysed for quantified dynamic behaviour in a number of different ways.

The language may be used to generate a system of ordinary differential equations (ODEs).

Deriving quantitative data

PEPA models can be analysed for quantified dynamic behaviour in a number of different ways.

The language may be used to generate a system of ordinary differential equations (ODEs).

PEPA MODEL —syntactic analysis→ ACTIVITY MATRIX —continuous interpretation→ ODEs

Each of these has tool support so that the underlying model is derived automatically according to the predefined rules.