

# SPA for quantitative analysis: Lecture 4 — Tackling State Space Explosion

Jane Hillston

LFCS, School of Informatics  
The University of Edinburgh  
Scotland

6th March 2013

# Outline

- 1 Introduction
- 2 Model reduction
  - Numerical representation
- 3 Decomposed solutions
- 4 Simulation
- 5 Fluid Approximation
- 6 Summary

# Outline

- 1 Introduction
- 2 Model reduction
  - Numerical representation
- 3 Decomposed solutions
- 4 Simulation
- 5 Fluid Approximation
- 6 Summary

# 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  $Q$ , and the  $N$ -dimensional probability vector  $\pi$ , 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.)

# 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  $Q$ , and the  $N$ -dimensional probability vector  $\pi$ , 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.)

# 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  $Q$ , and the  $N$ -dimensional probability vector  $\pi$ , 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

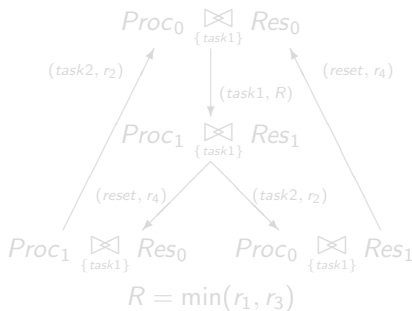
$$Proc_0 \stackrel{def}{=} (task1, r_1).Proc_1$$

$$Proc_1 \stackrel{def}{=} (task2, r_2).Proc_0$$

$$Res_0 \stackrel{def}{=} (task1, r_3).Res_1$$

$$Res_1 \stackrel{def}{=} (reset, r_4).Res_0$$

$$Proc_0 \boxtimes_{\{task1\}} Res_0$$



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

# A simple example: processors and resources

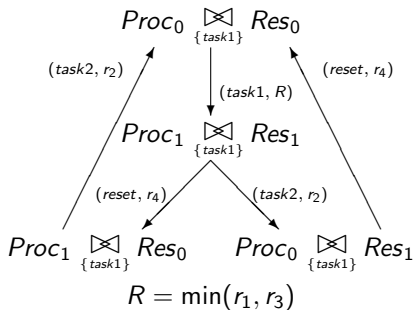
$$Proc_0 \stackrel{def}{=} (task1, r_1).Proc_1$$

$$Proc_1 \stackrel{def}{=} (task2, r_2).Proc_0$$

$$Res_0 \stackrel{def}{=} (task1, r_3).Res_1$$

$$Res_1 \stackrel{def}{=} (reset, r_4).Res_0$$

$$Proc_0 \boxtimes_{\{task1\}} Res_0$$



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



# A simple example: processors and resources

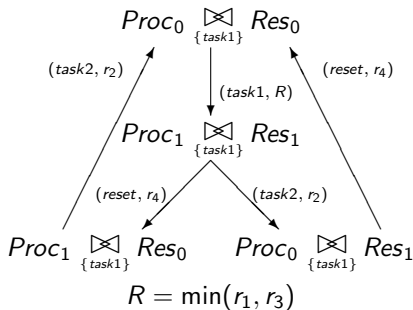
$$Proc_0 \stackrel{def}{=} (task1, r_1).Proc_1$$

$$Proc_1 \stackrel{def}{=} (task2, r_2).Proc_0$$

$$Res_0 \stackrel{def}{=} (task1, r_3).Res_1$$

$$Res_1 \stackrel{def}{=} (reset, r_4).Res_0$$

$$Proc_0 \begin{array}{c} \boxtimes \\ \{task1\} \end{array} Res_0$$



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

# Simple example : multiple instances

$$Proc_0 \stackrel{def}{=} (task1, r_1).Proc_1$$

$$Proc_1 \stackrel{def}{=} (task2, r_2).Proc_0$$

$$Res_0 \stackrel{def}{=} (task1, r_3).Res_1$$

$$Res_1 \stackrel{def}{=} (reset, r_4).Res_0$$

$$Proc_0[N_P] \boxtimes_{\{task1\}} Res_0[N_R]$$

## 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:  $2^{N_P} \times 2^{N_R}$ .

# Simple example : multiple instances

$$Proc_0 \stackrel{def}{=} (task1, r_1).Proc_1$$

$$Proc_1 \stackrel{def}{=} (task2, r_2).Proc_0$$

$$Res_0 \stackrel{def}{=} (task1, r_3).Res_1$$

$$Res_1 \stackrel{def}{=} (reset, r_4).Res_0$$

$$Proc_0[N_P] \boxtimes_{\{task1\}} Res_0[N_R]$$

## 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:  $2^{N_P} \times 2^{N_R}$ .

# Simple example : multiple instances

$$Proc_0 \stackrel{def}{=} (task1, r_1).Proc_1$$

$$Proc_1 \stackrel{def}{=} (task2, r_2).Proc_0$$

$$Res_0 \stackrel{def}{=} (task1, r_3).Res_1$$

$$Res_1 \stackrel{def}{=} (reset, r_4).Res_0$$

$$Proc_0[N_P] \boxtimes_{\{task1\}} Res_0[N_R]$$

## 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.
- 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;
  - decomposed solution techniques
  - stochastic simulation over the discrete state space;
  - fluid approximation of the 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](#);
  - [decomposed solution](#) techniques
  - [stochastic simulation](#) over the discrete state space;
  - [fluid approximation](#) of the 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](#);
  - [decomposed solution](#) techniques
  - [stochastic simulation](#) over the discrete state space;
  - [fluid approximation](#) of the 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](#);
  - [decomposed solution](#) techniques
  - [stochastic simulation](#) over the discrete state space;
  - [fluid approximation](#) of the 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](#);
  - [decomposed solution](#) techniques
  - [stochastic simulation](#) over the discrete state space;
  - [fluid approximation](#) of the 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](#);
  - [decomposed solution](#) techniques
  - [stochastic simulation](#) over the discrete state space;
  - [fluid approximation](#) of the state space.

# Outline

- 1 Introduction
- 2 Model reduction
  - Numerical representation
- 3 Decomposed solutions
- 4 Simulation
- 5 Fluid Approximation
- 6 Summary

# 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**.

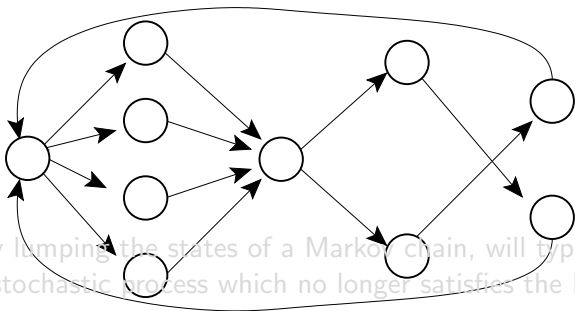
# Aggregation and lumpability

- **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**.

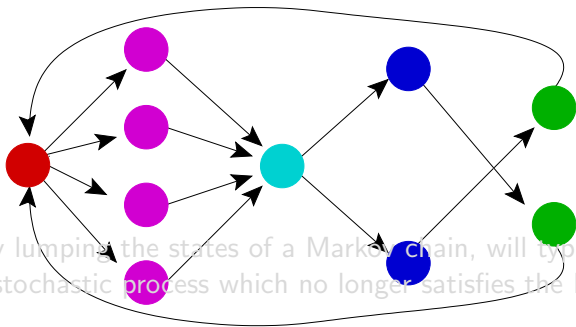
# Aggregation and lumpability

- **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



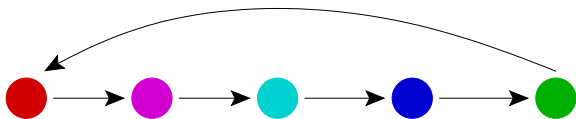
# Reducing by lumpability



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

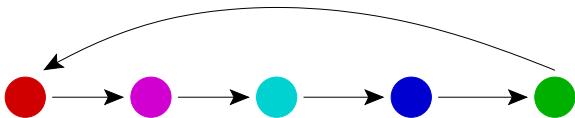


# Reducing by lumpability



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

# 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.

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**.

# 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**.

# 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**.

# Losing identity

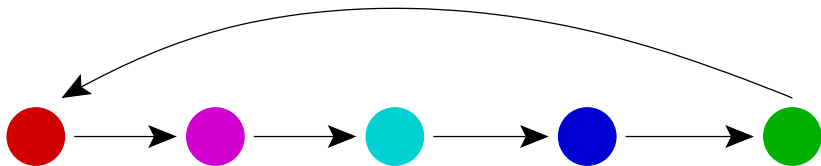
The **syntactic** nature of PEPA makes models easily understood by humans, but not so convenient for approaches such as aggregation and simulation.

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**.

# 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

$$Proc_0 \stackrel{def}{=} (task1, r_1).Proc_1$$

$$Proc_1 \stackrel{def}{=} (task2, r_2).Proc_0$$

$$Res_0 \stackrel{def}{=} (task1, r_1).Res_1$$

$$Res_1 \stackrel{def}{=} (reset, s).Res_0$$

$$(Res_0 \parallel Res_0) \bowtie_{\{task1\}} (Proc_0 \parallel Proc_0)$$



# Numerical vector form

For our example model:

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

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.

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

# 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.

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

# 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

$(Res_0, Res_1, Proc_0, Proc_1)$

$(Res_1, Res_0, Proc_0, Proc_1)$

$(Res_0, Res_1, Proc_1, Proc_0)$

$(Res_1, Res_0, Proc_1, Proc_0)$

# 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

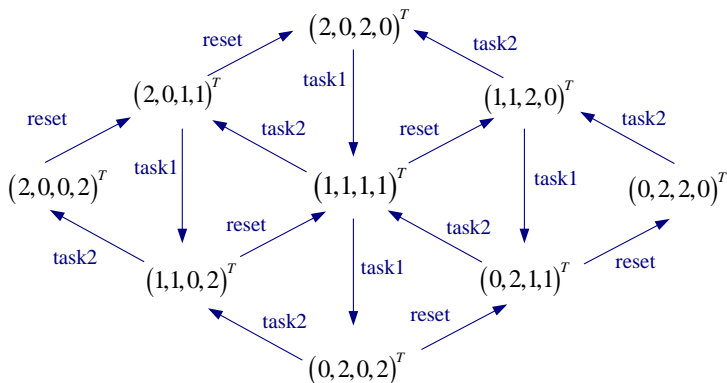
$(Res_0, Res_1, Proc_0, Proc_1)$

$(Res_1, Res_0, Proc_0, Proc_1)$

$(Res_0, Res_1, Proc_1, Proc_0)$

$(Res_1, Res_0, Proc_1, Proc_0)$

# 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.

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.

# 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.

# 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.



# 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.

## Using this result in practice

There are well-known algorithms such as [Paige and Tarjan](#) for finding the maximal partition of a graph according to some equivalence.

However in practice we would much rather construct the aggregated state space directly.

The first approach to this used [canonical forms](#) but still worked syntactically to identify states. [Gilmore, Hillston and Ribaud, IEEE TSE 2001].

A more recent approach shifts to a [numerical representation of states and transitions](#). [Jie Ding, PhD thesis, Edin. 2010]

## Using this result in practice

There are well-known algorithms such as [Paige and Tarjan](#) for finding the maximal partition of a graph according to some equivalence.

However in practice we would much rather construct the aggregated state space directly.

The first approach to this used [canonical forms](#) but still worked syntactically to identify states. [Gilmore, Hillston and Ribaud, IEEE TSE 2001].

A more recent approach shifts to a [numerical representation of states and transitions](#). [Jie Ding, PhD thesis, Edin. 2010]

## Using this result in practice

There are well-known algorithms such as [Paige and Tarjan](#) for finding the maximal partition of a graph according to some equivalence.

However in practice we would much rather construct the aggregated state space directly.

The first approach to this used [canonical forms](#) but still worked syntactically to identify states. [Gilmore, Hillston and Ribaud, IEEE TSE 2001].

A more recent approach shifts to a [numerical representation of states and transitions](#). [Jie Ding, PhD thesis, Edin. 2010]

## Using this result in practice

There are well-known algorithms such as [Paige and Tarjan](#) for finding the maximal partition of a graph according to some equivalence.

However in practice we would much rather construct the aggregated state space directly.

The first approach to this used [canonical forms](#) but still worked syntactically to identify states. [Gilmore, Hillston and Ribaud, IEEE TSE 2001].

A more recent approach shifts to a [numerical representation of states and transitions](#). [Jie Ding, PhD thesis, Edin. 2010]

# Achieving aggregation

- To overcome state-space explosion problem encountered in performance analysis, many mathematical tools and approaches have been proposed.
- **Syntactic** nature of PEPA (as well as other SPAs) makes models easily understood by humans, but not so convenient for computers to directly apply these tools and approaches.
- By shifting to a *numerical state representation* we can more readily exploit results such as aggregation and access mathematical interpretations (i.e. **fluid approximation**).

# Numerical Vector Form [QEST 2005]

## Definition

For an arbitrary PEPA model  $\mathcal{M}$  with  $n$  component types  $C_i, i = 1, 2, \dots, n$ , each with  $d_i$  distinct local derivatives, the numerical vector form of  $\mathcal{M}$ ,  $\mathbf{m}(\mathcal{M})$ , is a vector with  $d = \sum_{i=1}^n d_i$  entries. The entry  $\mathbf{m}[C_j]$  records how many instances of the  $j$ th local derivative of component type  $C_i$  are exhibited in the current state.

The entries in the system vector or a sequential component's vector are no longer syntactic terms representing the local derivative, but the number of components currently exhibiting this local derivative.

# Activity matrix

If the states are captured in a numerical form, it is appropriate to also capture the transitions between them numerically.

The transitions arise from the activities in the PEPA model. Thus we use the syntax of the process algebra to infer the impact of each activity on the current state, numerically.

This information is represented in the [activity matrix](#).



# Activity matrix

If the states are captured in a numerical form, it is appropriate to also capture the transitions between them numerically.

The transitions arise from the activities in the PEPA model. Thus we use the syntax of the process algebra to infer the impact of each activity on the current state, numerically.

This information is represented in the [activity matrix](#).

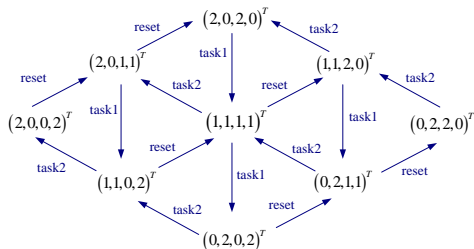
# Activity matrix

If the states are captured in a numerical form, it is appropriate to also capture the transitions between them numerically.

The transitions arise from the activities in the PEPA model. Thus we use the syntax of the process algebra to infer the impact of each activity on the current state, numerically.

This information is represented in the [activity matrix](#).

# Activity matrix



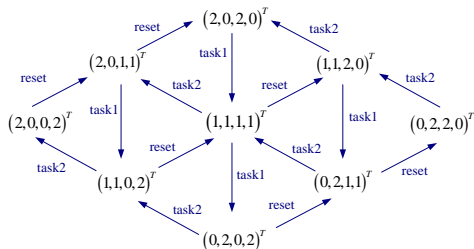
	$I^{task_1}$	$I^{task_2}$	$I^{reset}$
$Proc_0$	-1	1	0
$Proc_1$	1	-1	0
$Res_0$	-1	0	1
$Res_1$	1	0	-1

Each activity corresponds a transition vector. All transition vectors form an *activity matrix*  $\mathbf{C}$ . For example,

$$(2, 0, 2, 0)^T + I^{task_1} = (1, 1, 1, 1)^T, \quad (1, 1, 1, 1)^T + I^{task_2} = (2, 0, 1, 1)^T.$$

$$(2, 0, 1, 1)^T + I^{reset} = (2, 0, 2, 0)^T.$$

# Activity matrix



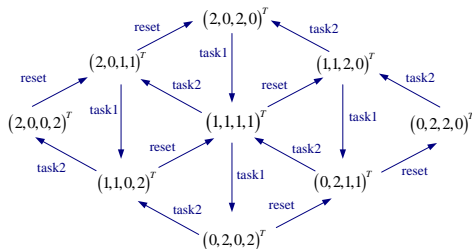
	$I^{task_1}$	$I^{task_2}$	$I^{reset}$
$Proc_0$	-1	1	0
$Proc_1$	1	-1	0
$Res_0$	-1	0	1
$Res_1$	1	0	-1

Each activity corresponds a transition vector. All transition vectors form an *activity matrix*  $\mathbf{C}$ . For example,

$$(2, 0, 2, 0)^T + I^{task_1} = (1, 1, 1, 1)^T, \quad (1, 1, 1, 1)^T + I^{task_2} = (2, 0, 1, 1)^T.$$

$$(2, 0, 1, 1)^T + I^{reset} = (2, 0, 2, 0)^T.$$

# Activity matrix



	$I^{task_1}$	$I^{task_2}$	$I^{reset}$
$Proc_0$	-1	1	0
$Proc_1$	1	-1	0
$Res_0$	-1	0	1
$Res_1$	1	0	-1

Each activity corresponds a transition vector. All transition vectors form an *activity matrix*  $\mathbf{C}$ . For example,

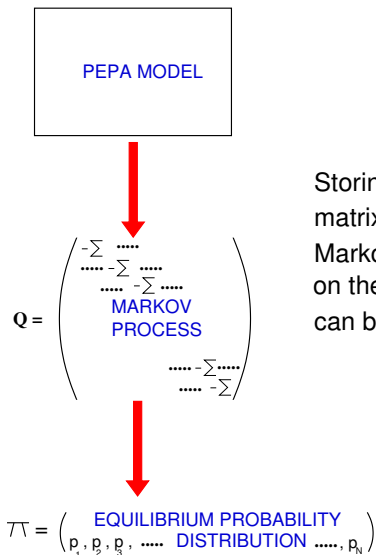
$$(2, 0, 2, 0)^T + I^{task_1} = (1, 1, 1, 1)^T, \quad (1, 1, 1, 1)^T + I^{task_2} = (2, 0, 1, 1)^T.$$

$$(2, 0, 1, 1)^T + I^{reset} = (2, 0, 2, 0)^T.$$

# Outline

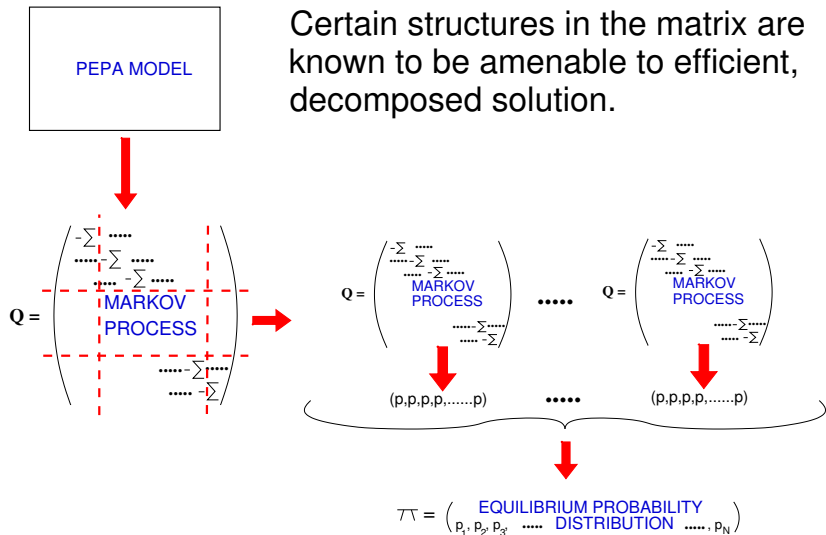
- 1 Introduction
- 2 Model reduction
  - Numerical representation
- 3 Decomposed solutions**
- 4 Simulation
- 5 Fluid Approximation
- 6 Summary

# Characterising efficient solution



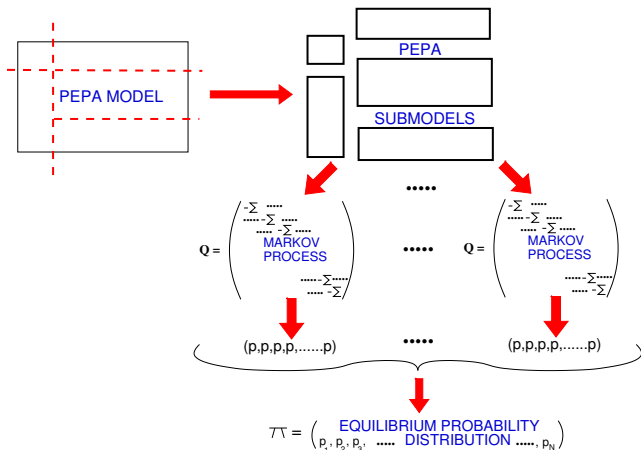
Storing and manipulating the matrix which represents the Markov process places limitations on the size of model which can be analysed.

# Characterising efficient solution



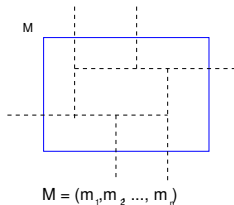


# Characterising efficient solution

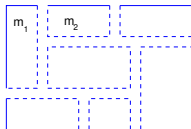


Finding the corresponding structures in the process algebra means that these techniques can be applied automatically, before the monolithic matrix is formed.

# Decomposed solution: product form models



Partition the model  $M$  into  $n$  **statistically independent** submodels  $m_1, m_2, \dots, m_n$



In isolation, find the steady state distribution  $p$  for each of the submodels  $m_i$

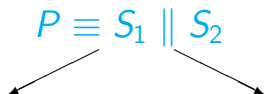
$p(M)$

$$p(M) = G \times p(m_1) \times p(m_2) \times \dots \times p(m_n)$$

Form the steady state distribution of  $M$  as the product of the solutions for each submodel  $m_i$  and a normalising constant

When do PEPA components behave as if they were statistically independent...?

# Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$


- Quasi-reversibility
- Reversibility
- Routing process approach

# Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted **direct interaction** between components with a particular structure

$$P \equiv S_1 \bowtie_L S_2$$

$S_1$ ,  $S_2$  and  $L$  all restricted

- Quasi-reversibility
- Reversibility
- Routing process approach

# Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted **direct interaction**  
between components with a  
particular structure

$$P \equiv S_1 \bowtie_L S_2$$

$S_1$ ,  $S_2$  and  $L$  all restricted

- Quasi-reversibility
- Reversibility
- Routing process approach

# Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted **direct interaction** between components with a particular structure

$$P \equiv S_1 \bowtie_L S_2$$

$S_1$ ,  $S_2$  and  $L$  all restricted

- Quasi-reversibility
- Reversibility
- Routing process approach

# Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted **direct interaction** between components with a particular structure

$$P \equiv S_1 \bowtie_L S_2$$

$S_1, S_2$  and  $L$  all restricted

- Quasi-reversibility
- Reversibility
- Routing process approach

# Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted **direct interaction** between components with a particular structure

$$P \equiv S_1 \boxtimes_L S_2$$

$S_1, S_2$  and  $L$  all restricted

- Quasi-reversibility
- Reversibility
- Routing process approach

Add **indirect interaction** via a third component with a particular structure and type of interaction

$$P \equiv (S_1 \parallel S_2) \boxtimes_L R$$

$L$  and  $R$  restricted (wrt  $S_1$  and  $S_2$ )

- Boucherie resource contention
- Queueing discipline models
- Quasi-separability



# Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted **direct interaction** between components with a particular structure

$$P \equiv S_1 \boxtimes_L S_2$$

$S_1, S_2$  and  $L$  all restricted

- Quasi-reversibility
- Reversibility
- Routing process approach

Add **indirect interaction** via a third component with a particular structure and type of interaction

$$P \equiv (S_1 \parallel S_2) \boxtimes_L R$$

$L$  and  $R$  restricted (wrt  $S_1$  and  $S_2$ )

- Boucherie resource contention
- Queueing discipline models
- Quasi-separability

# Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted **direct interaction** between components with a particular structure

$$P \equiv S_1 \boxtimes_L S_2$$

$S_1, S_2$  and  $L$  all restricted

- Quasi-reversibility
- Reversibility
- Routing process approach

Add **indirect interaction** via a third component with a particular structure and type of interaction

$$P \equiv (S_1 \parallel S_2) \boxtimes_L R$$

$L$  and  $R$  restricted (wrt  $S_1$  and  $S_2$ )

- Boucherie resource contention
- Queueing discipline models
- Quasi-separability

# Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted **direct interaction** between components with a particular structure

$$P \equiv S_1 \bowtie_L S_2$$

$S_1, S_2$  and  $L$  all restricted

- Quasi-reversibility
- Reversibility
- Routing process approach

Add **indirect interaction** via a third component with a particular structure and type of interaction

$$P \equiv (S_1 \parallel S_2) \bowtie_L R$$

$L$  and  $R$  restricted (wrt  $S_1$  and  $S_2$ )

- Boucherie resource contention
- Queueing discipline models
- Quasi-separability

# Approximate solutions

Numerical solution of the full Markov process is regarded as the **exact** result and aggregation based on lumpability is also regarded as being exact.

However due to the difficulties of staying within the confines of the Markov property most techniques for tackling state space explosion are not exact.

We are sometimes prepared to **trade exactness for tractability**.

# Approximate solutions

Numerical solution of the full Markov process is regarded as the **exact** result and aggregation based on lumpability is also regarded as being exact.

However due to the difficulties of staying within the confines of the Markov property most techniques for tackling state space explosion are not exact.

We are sometimes prepared to **trade exactness for tractability**.

# Approximate solutions

Numerical solution of the full Markov process is regarded as the **exact** result and aggregation based on lumpability is also regarded as being exact.

However due to the difficulties of staying within the confines of the Markov property most techniques for tackling state space explosion are not exact.

We are sometimes prepared to **trade exactness for tractability**.

# Other decomposition results

Other forms of decomposed solution of SPA models have also been developed which give approximate results:

**Time Scale Decomposition:** Based on a well-established technique in the underlying CTMC, this approach partitions the states of the process according to the rates at which they undertake activities. Fast interacting states are modelled in detail in isolation, and an aggregated model captures the transitions between the clusters of states.

**Throughput Approximation:** The model is partitioned into two in such a way that there is a one flow each way between them. Each is solved to give an estimate of the flow into the other and alternate solutions are iterated until convergence.

# Other decomposition results

Other forms of decomposed solution of SPA models have also been developed which give approximate results:

**Time Scale Decomposition:** Based on a well-established technique in the underlying CTMC, this approach partitions the states of the process according to the rates at which they undertake activities. Fast interacting states are modelled in detail in isolation, and an aggregated model captures the transitions between the clusters of states.

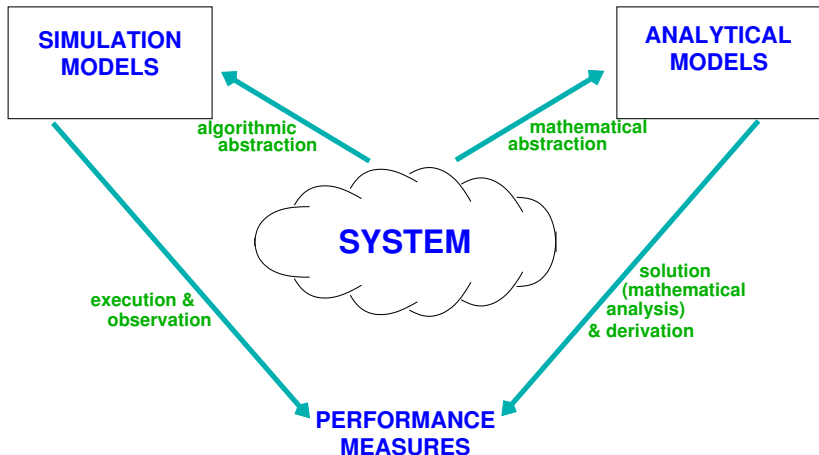
**Throughput Approximation:** The model is partitioned into two in such a way that there is a one flow each way between them. Each is solved to give an estimate of the flow into the other and alternate solutions are iterated until convergence.



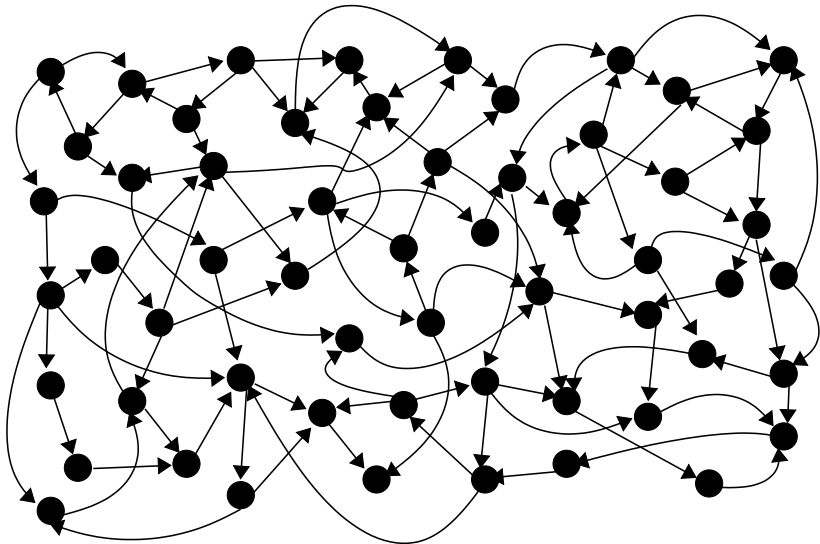
# Outline

- 1 Introduction
- 2 Model reduction
  - Numerical representation
- 3 Decomposed solutions
- 4 Simulation**
- 5 Fluid Approximation
- 6 Summary

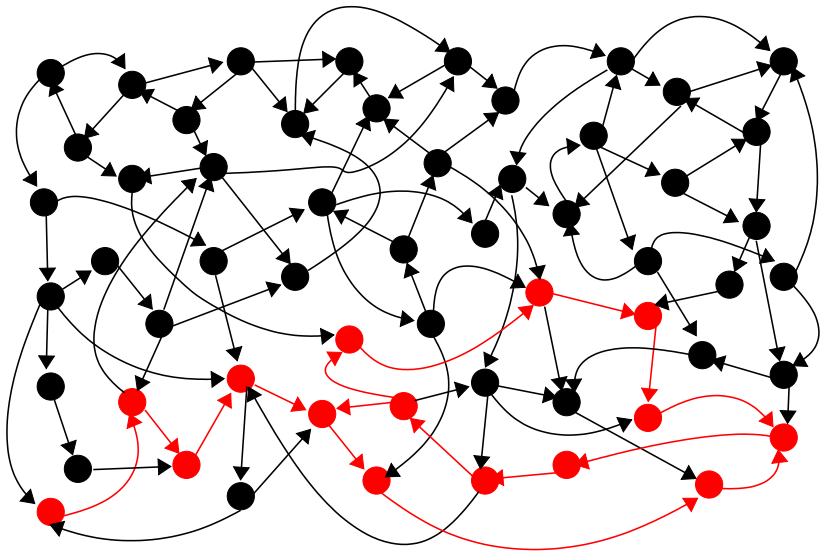
# Introduction



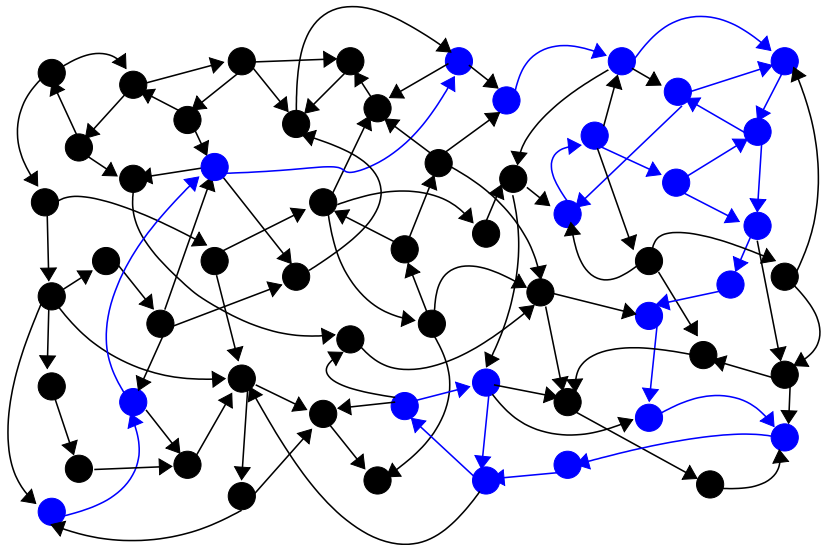
# State space and sample paths



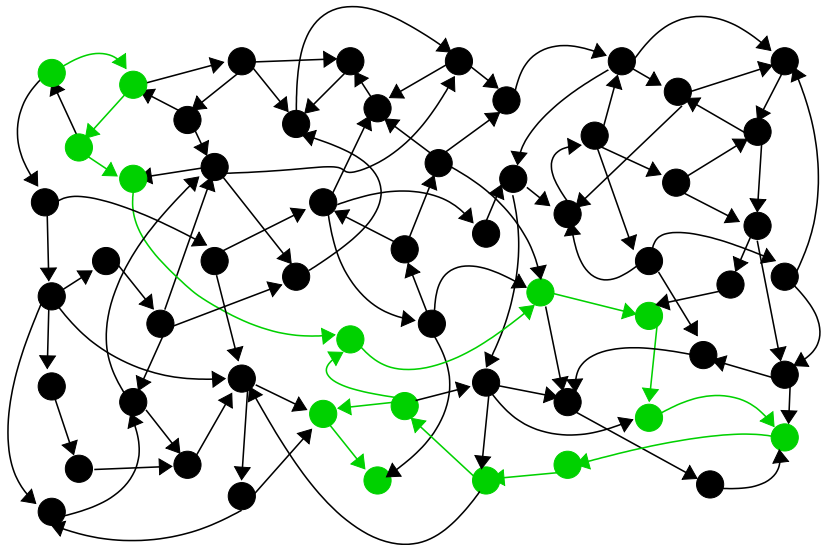
# State space and sample paths



# State space and sample paths



# State space and sample paths



# Sample paths and runs

- Using the analytic approach of Markov processes we characterised **all possible sample paths** by the global balance equations.
- Using simulation we investigate the sample paths directly.
- We allow the model to trace out a sample path over the state space.
- Each **run** of the simulation model will generate another, usually distinct, sample path.

# Sample paths and runs

- Using the analytic approach of Markov processes we characterised **all possible sample paths** by the global balance equations.
- Using simulation we investigate the sample paths directly.
- We allow the model to trace out a sample path over the state space.
- Each **run** of the simulation model will generate another, usually distinct, sample path.



# Sample paths and runs

- Using the analytic approach of Markov processes we characterised **all possible sample paths** by the global balance equations.
- Using simulation we investigate the sample paths directly.
- We allow the model to trace out a sample path over the state space.
- Each **run** of the simulation model will generate another, usually distinct, sample path.

# Sample paths and runs

- Using the analytic approach of Markov processes we characterised **all possible sample paths** by the global balance equations.
- Using simulation we investigate the sample paths directly.
- We allow the model to trace out a sample path over the state space.
- Each **run** of the simulation model will generate another, usually distinct, sample path.

# 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.

# 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.

# 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.

# 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

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.

# The Gillespie Stochastic Simulation Algorithm

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  $(\alpha_1, r_1), (\alpha_2, r_2), \dots, (\alpha_n, r_n)$  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**  $r_1 + r_2 + \dots + r_n$ .

We also know that the probability that it is the activity of type  $\alpha_i$  is

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

# Two Observations

If we have a number of possible activities  $(\alpha_1, r_1), (\alpha_2, r_2), \dots, (\alpha_n, r_n)$  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**  $r_1 + r_2 + \dots + r_n$ .

We also know that the probability that it is the activity of type  $\alpha_i$  is

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

# 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.

# The Gillespie Stochastic Simulation Algorithm

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

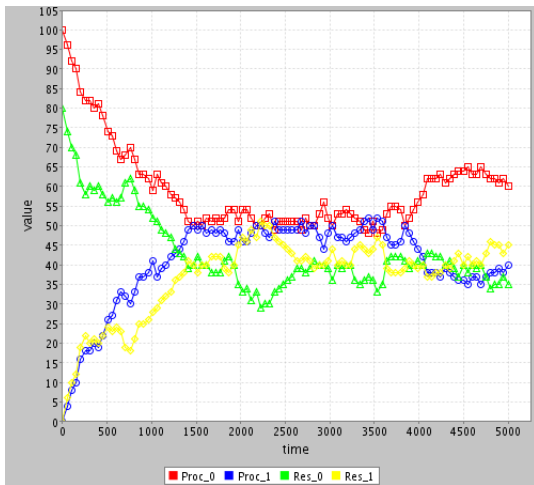
- the first determines the delay until the next activity completes,
- the second determines which activity that will be.

# The Gillespie Stochastic Simulation Algorithm

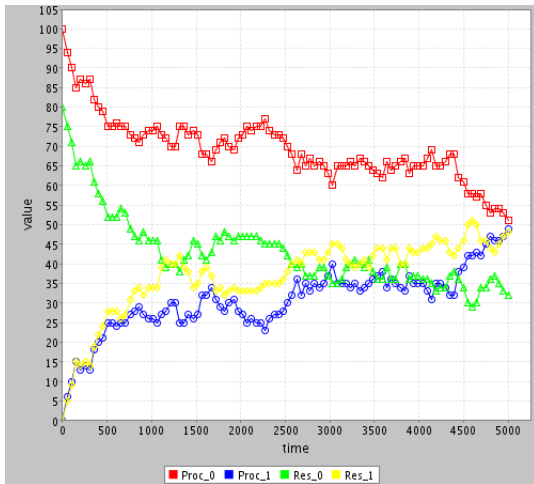
Thus we need only draw two random numbers for each step of the simulation algorithm:

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

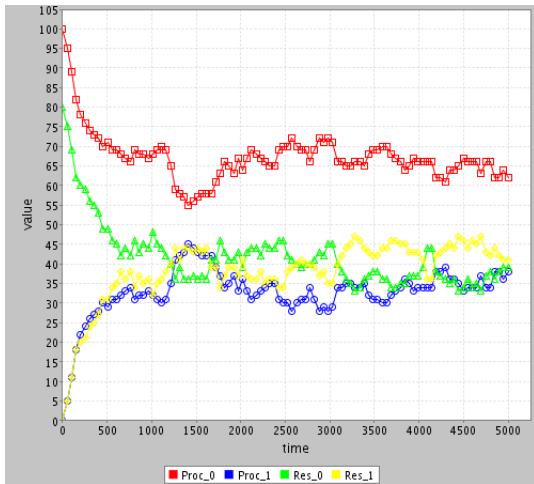
# 100 processors and 80 resources (simulation run A)



# 100 processors and 80 resources (simulation run B)

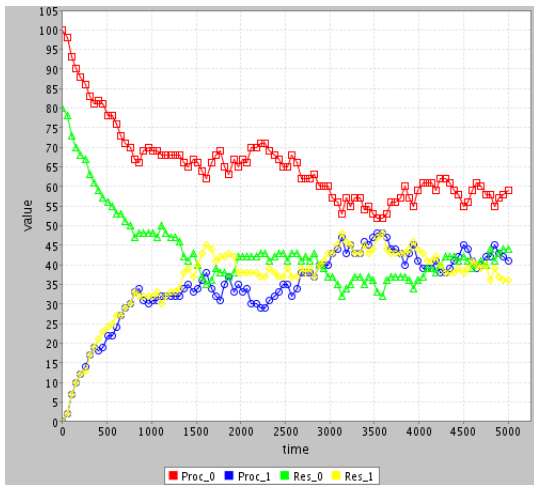


# 100 processors and 80 resources (simulation run C)

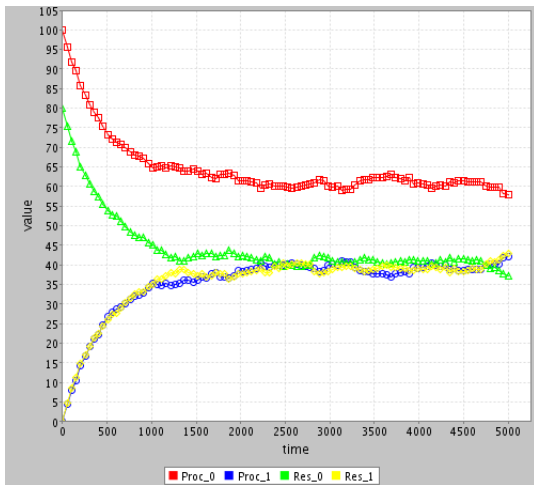




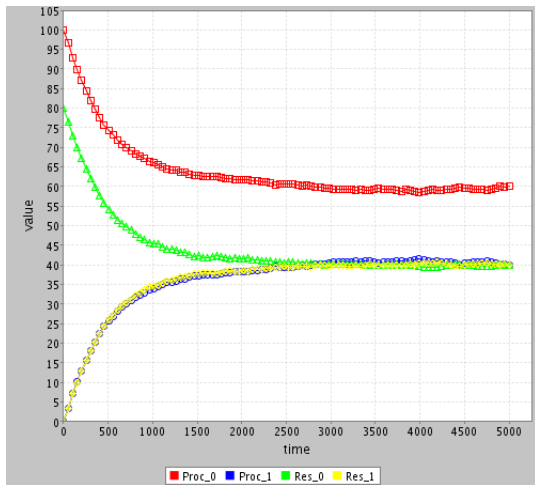
# 100 processors and 80 resources (simulation run D)



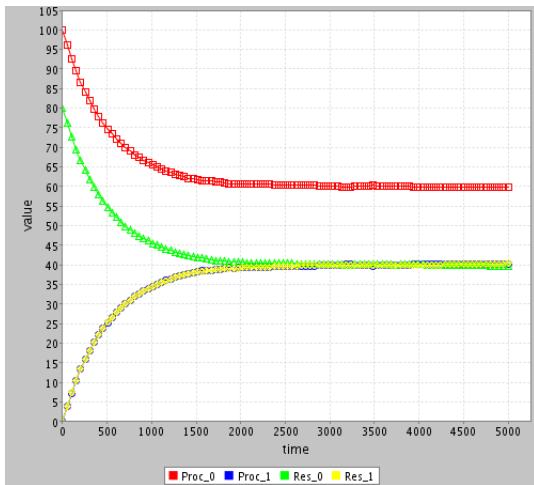
# 100 processors and 80 resources (average of 10 runs)



# 100 Processors and 80 resources (average of 100 runs)



# 100 processors and 80 resources (average of 1000 runs)



# Outline

- 1 Introduction
- 2 Model reduction
  - Numerical representation
- 3 Decomposed solutions
- 4 Simulation
- 5 Fluid Approximation**
- 6 Summary

# Fluid Approximation

The fourth 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**.

# Fluid Approximation

The fourth 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.

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 expectation it becomes easier to interpret.



# 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 expectation it becomes easier to interpret.

# Fluid approximation

- In a PEPA model the state at any current time is the local derivative or **state of each component** of the model.
- We can represent the state of the system as the **count** of the current number of each possible local derivative or **component type**.
- We can **approximate** the behaviour of the model by treating each count as a **continuous variable**, and the state of the model as a whole as the set of such variables.
- The **evolution** of each count variable can then be described by an **ordinary differential equation**

# Fluid approximation

- In a PEPA model the state at any current time is the local derivative or **state of each component** of the model.
- We can represent the state of the system as the **count** of the current number of each possible local derivative or **component type**.
- We can **approximate** the behaviour of the model by treating each count as a **continuous variable**, and the state of the model as a whole as the set of such variables.
- The **evolution** of each count variable can then be described by an **ordinary differential equation** (**assuming rates are deterministic**).

# Fluid approximation

- In a PEPA model the state at any current time is the local derivative or **state of each component** of the model.
- We can represent the state of the system as the **count** of the current number of each possible local derivative or **component type**.
- We can **approximate** the behaviour of the model by treating each count as a **continuous variable**, and the state of the model as a whole as the set of such variables.
- The **evolution** of each count variable can then be described by an **ordinary differential equation** (**assuming rates are deterministic**).

# Fluid approximation

- In a PEPA model the state at any current time is the local derivative or **state of each component** of the model.
- We can represent the state of the system as the **count** of the current number of each possible local derivative or **component type**.
- We can **approximate** the behaviour of the model by treating each count as a **continuous variable**, and the state of the model as a whole as the set of such variables.
- The **evolution** of each count variable can then be described by an **ordinary differential equation** (**assuming rates are deterministic**).

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

$$Proc_0 \stackrel{def}{=} (task1, r_1).Proc_1$$

$$Proc_1 \stackrel{def}{=} (task2, r_2).Proc_0$$

$$Res_0 \stackrel{def}{=} (task1, r_1).Res_1$$

$$Res_1 \stackrel{def}{=} (reset, r_4).Res_0$$

$$Proc_0[N_P] \begin{array}{c} \boxtimes \\ \{task1\} \end{array} Res_0[N_R]$$

We can capture exactly this relationship between activities and components the **activity matrix** which has one row for each component and one column for each activity.

# Example revisited

$$Proc_0 \stackrel{def}{=} (task1, r_1).Proc_1$$

$$Proc_1 \stackrel{def}{=} (task2, r_2).Proc_0$$

$$Res_0 \stackrel{def}{=} (task1, r_1).Res_1$$

$$Res_1 \stackrel{def}{=} (reset, r_4).Res_0$$

$$Proc_0[N_P] \boxtimes_{\{task1\}} Res_0[N_R]$$

- *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 exactly this relationship between activities and components the **activity matrix** which has one row for each component and one column for each activity.



# Example revisited

$$Proc_0 \stackrel{def}{=} (task1, r_1).Proc_1$$

$$Proc_1 \stackrel{def}{=} (task2, r_2).Proc_0$$

$$Res_0 \stackrel{def}{=} (task1, r_1).Res_1$$

$$Res_1 \stackrel{def}{=} (reset, r_4).Res_0$$

$$Proc_0[N_P] \boxtimes_{\{task1\}} Res_0[N_R]$$

ODE interpretation

$$\frac{dx_1}{dt} = -r_1 \min(x_1, x_3) + r_2 x_2$$

$x_1 =$  no. of  $Proc_1$

$$\frac{dx_2}{dt} = r_1 \min(x_1, x_3) - r_2 x_2$$

$x_2 =$  no. of  $Proc_2$

$$\frac{dx_3}{dt} = -r_1 \min(x_1, x_3) + r_4 x_4$$

$x_3 =$  no. of  $Res_0$

$$\frac{dx_4}{dt} = r_1 \min(x_1, x_3) - r_4 x_4$$

$x_4 =$  no. of  $Res_1$

We can capture exactly this relationship between activities and components the **activity matrix** which has one row for each component and one column for each activity.

# Differential equations from PEPA models

- As we have already seen in deriving the activity matrix, 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.

# Differential equations from PEPA models

Let  $N(\mathcal{C}_{ij}, t)$  denote the number of  $\mathcal{C}_{ij}$  type components at time  $t$ .

# Differential equations from PEPA models

Let  $N(C_{ij}, t)$  denote the number of  $C_{ij}$  type components at time  $t$ .

Consider the change in a small time  $\delta t$ :

$$\begin{aligned}
 N(C_{ij}, t + \delta t) - N(C_{ij}, t) = & \\
 & - \underbrace{\sum_{(\alpha,r) \in \text{Ex}(C_{ij})} r \times \min_{C_{k_l} \in \text{pre}(\alpha,r)} (N(C_{k_l}, t)) \delta t}_{\text{exit activities}} \\
 & + \underbrace{\sum_{(\alpha,r) \in \text{En}(C_{ij})} r \times \min_{C_{k_l} \in \text{pre}(\alpha,r)} (N(C_{k_l}, t)) \delta t}_{\text{entry activities}}
 \end{aligned}$$

# Differential equations from PEPA models

Let  $N(C_{ij}, t)$  denote the number of  $C_{ij}$  type components at time  $t$ .

Consider the change in a small time  $\delta t$ :

$$\begin{aligned}
 N(C_{ij}, t + \delta t) - N(C_{ij}, t) = & \\
 & - \underbrace{\sum_{(\alpha,r) \in \text{Ex}(C_{ij})} r \times \min_{C_{k_l} \in \text{pre}(\alpha,r)} (N(C_{k_l}, t)) \delta t}_{\text{exit activities}} \\
 & + \underbrace{\sum_{(\alpha,r) \in \text{En}(C_{ij})} r \times \min_{C_{k_l} \in \text{pre}(\alpha,r)} (N(C_{k_l}, t)) \delta t}_{\text{entry activities}}
 \end{aligned}$$

# Differential equations from PEPA models

Let  $N(C_{ij}, t)$  denote the number of  $C_{ij}$  type components at time  $t$ .

Consider the change in a small time  $\delta t$ :

$$\begin{aligned}
 N(C_{ij}, t + \delta t) - N(C_{ij}, t) = & \\
 & - \underbrace{\sum_{(\alpha,r) \in \text{Ex}(C_{ij})} r \times \min_{C_{k_l} \in \text{pre}(\alpha,r)} (N(C_{k_l}, t)) \delta t}_{\text{exit activities}} \\
 & + \underbrace{\sum_{(\alpha,r) \in \text{En}(C_{ij})} r \times \min_{C_{k_l} \in \text{pre}(\alpha,r)} (N(C_{k_l}, t)) \delta t}_{\text{entry activities}}
 \end{aligned}$$

# Differential equations from PEPA models

Let  $N(C_{ij}, t)$  denote the number of  $C_{ij}$  type components at time  $t$ .

Consider the change in a small time  $\delta t$ :

$$\begin{aligned}
 N(C_{ij}, t + \delta t) - N(C_{ij}, t) = & \\
 & - \underbrace{\sum_{(\alpha,r) \in \text{Ex}(C_{ij})} r \times \min_{C_{k_l} \in \text{pre}(\alpha,r)} (N(C_{k_l}, t)) \delta t}_{\text{exit activities}} \\
 & + \underbrace{\sum_{(\alpha,r) \in \text{En}(C_{ij})} r \times \min_{C_{k_l} \in \text{pre}(\alpha,r)} (N(C_{k_l}, t)) \delta t}_{\text{entry activities}}
 \end{aligned}$$

# Differential equations from PEPA models

Let  $N(C_{ij}, t)$  denote the number of  $C_{ij}$  type components at time  $t$ .

Consider the change in a small time  $\delta t$ :

$$\begin{aligned}
 N(C_{ij}, t + \delta t) - N(C_{ij}, t) = & \\
 & - \underbrace{\sum_{(\alpha,r) \in Ex(C_{ij})} r \times \min_{C_{k_l} \in pre(\alpha,r)} (N(C_{k_l}, t)) \delta t}_{\text{exit activities}} \\
 & + \underbrace{\sum_{(\alpha,r) \in En(C_{ij})} r \times \min_{C_{k_l} \in pre(\alpha,r)} (N(C_{k_l}, t)) \delta t}_{\text{entry activities}}
 \end{aligned}$$



# Differential equations from PEPA models

Let  $N(C_{ij}, t)$  denote the number of  $C_{ij}$  type components at time  $t$ .

Dividing by  $\delta t$  and taking the limit,  $\delta t \rightarrow 0$ :

$$\begin{aligned} \frac{dN(C_{ij}, t)}{dt} = & - \sum_{(\alpha, r) \in Ex(C_{ij})} r \times \min_{C_{k_l} \in pre(\alpha, r)} (N(C_{k_l}, t)) \\ & + \sum_{(\alpha, r) \in En(C_{ij})} r \times \min_{C_{k_l} \in pre(\alpha, r)} (N(C_{k_l}, t)) \end{aligned}$$

# 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.
- 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.

# 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.

# 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

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

- 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.

## ODEs

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

- 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  $Proc_0[100] \bowtie_{\{task1\}} Res_0[80]$ .

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.

# Initialising the ODEs

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

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.

# Initialising the ODEs

Consider the model  $Proc_0[100] \begin{array}{c} \boxtimes \\ \{task1\} \end{array} Res_0[80]$ .

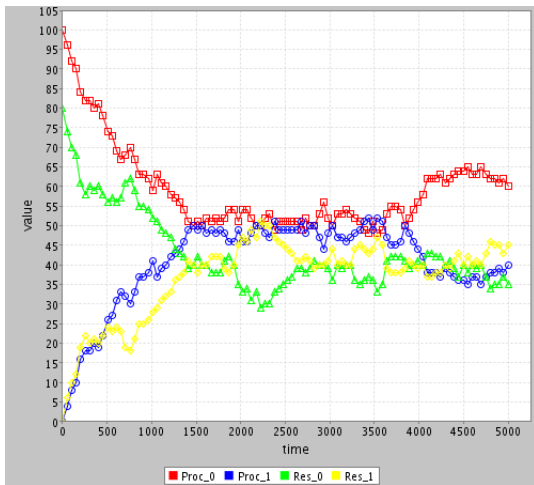
There are initially 100 processors, all starting in state  $Proc_0$  and 80 resources, all of which start in state  $Res_0$ .

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

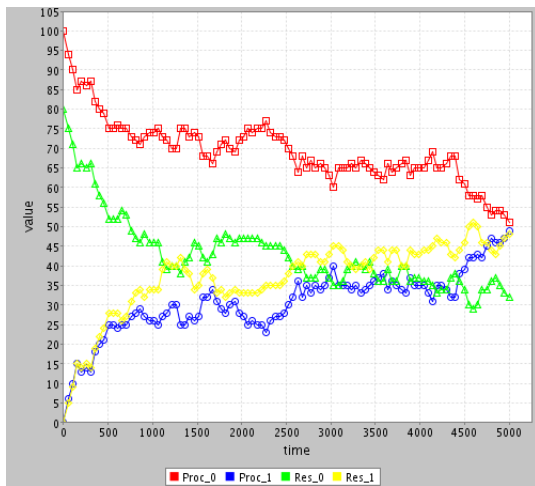
$$x_1(0) = 100 \quad x_2(0) = 0 \quad x_3(0) = 80 \quad x_4(0) = 0$$

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

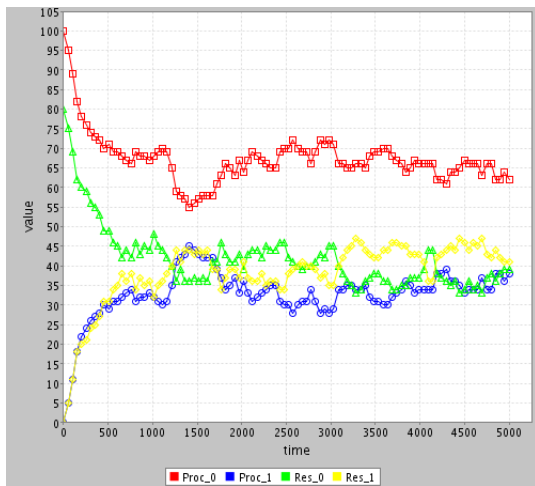
# 100 processors and 80 resources (simulation run A)



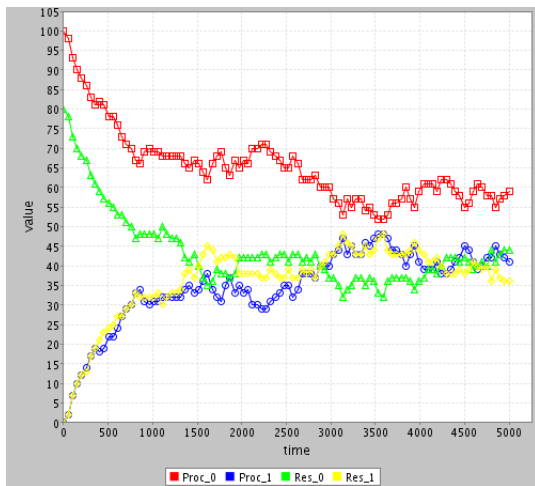
## 100 processors and 80 resources (simulation run B)



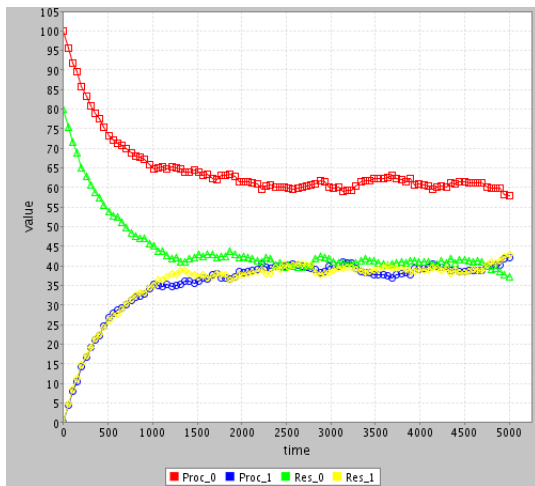
## 100 processors and 80 resources (simulation run C)



# 100 processors and 80 resources (simulation run D)

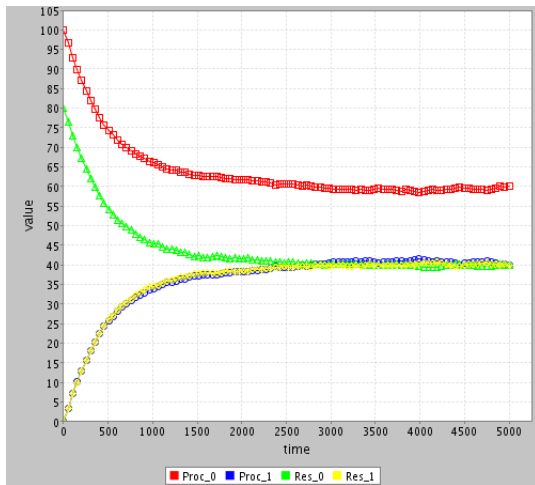


# 100 processors and 80 resources (average of 10 runs)

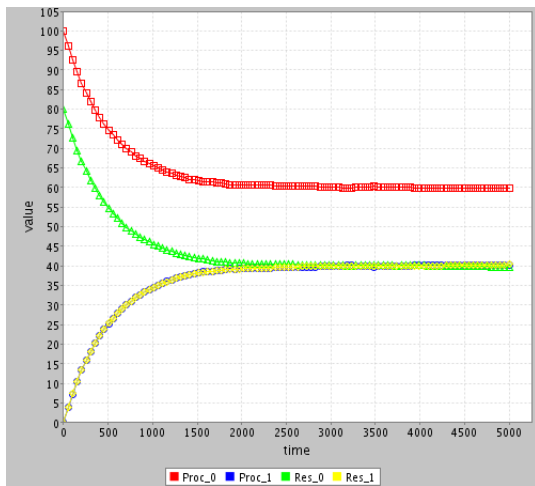




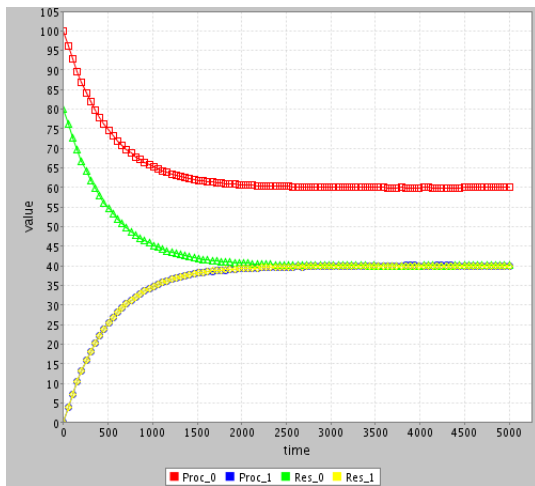
# 100 Processors and 80 resources (average of 100 runs)



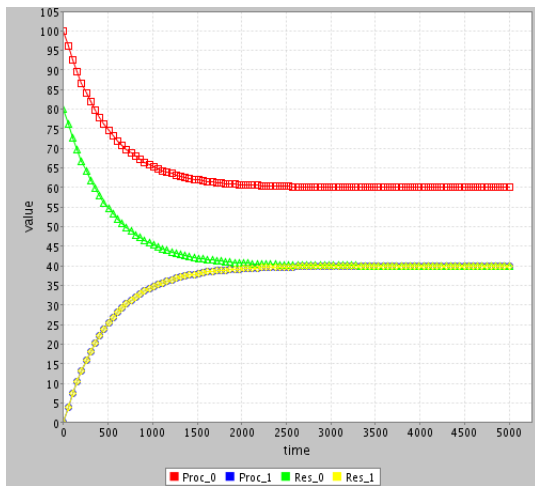
## 100 processors and 80 resources (average of 1000 runs)



## 100 processors and 80 resources (average of 10000 runs)



# 100 processors and 80 resources (ODE solution)



# Outline

- 1 Introduction
- 2 Model reduction
  - Numerical representation
- 3 Decomposed solutions
- 4 Simulation
- 5 Fluid Approximation
- 6 Summary**

# 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 \(CTMC\)](#).





# 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)**.



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

# References

- [Product form results for SPA](#) have been developed by Clark, Harrison, Hillston, Thomas and Sereno. A survey of some of the earlier results can be found in the tutorial chapter *Exploiting Structure in Solution: Decomposing Composed Models*, FMFA Lecture Notes, Springer Verlag [60].
- [Decomposition techniques](#) are also summarised in the above chapter but the definitive reference is Vassilis Mertsiotakis's PhD thesis from the University of Erlangen, 1997.
- [Numerical representation, activity matrix and fluid approximation](#) first appeared in the paper *Fluid flow approximation of PEPA models*, Proc. of 2nd Int. Conf. on the Quantitative Evaluation of Systems, September 2005. IEEE Computer Society Press [116].
- [Labelled activity matrix, results about relationship between steady state in ODEs and CTMC](#) are contained in Jie Ding's PhD thesis, University of Edinburgh 2010.