Process algebras for quantitative analysis: Lecture 4 — Tackling state space explosion

Jane Hillston

School of Informatics
The University of Edinburgh
Scotland

11th February 2010

Outline

Outline

Performance evaluation

Performance modellers usually make a choice between:

- Closed form analytical models (queueing networks);

Performance evaluation

Performance modellers usually make a choice between:

- Closed form analytical models (queueing networks);
- Discrete event simulations; or

Performance evaluation

Performance modellers usually make a choice between:

- Closed form analytical models (queueing networks);
- Discrete event simulations; or
- Numerical solution of continuous time Markov chains (CTMC) (Stochastic Petri nets, Stochastic process algebras etc.)

Performance evaluation

Performance modellers usually make a choice between:

- Closed form analytical models (queueing networks);
- Discrete event simulations; or
- Numerical solution of continuous time Markov chains (CTMC) (Stochastic Petri nets, Stochastic process algebras etc.)

Each of these models is based on a discrete state space and continuous time.

Performance evaluation

Performance modellers usually make a choice between:

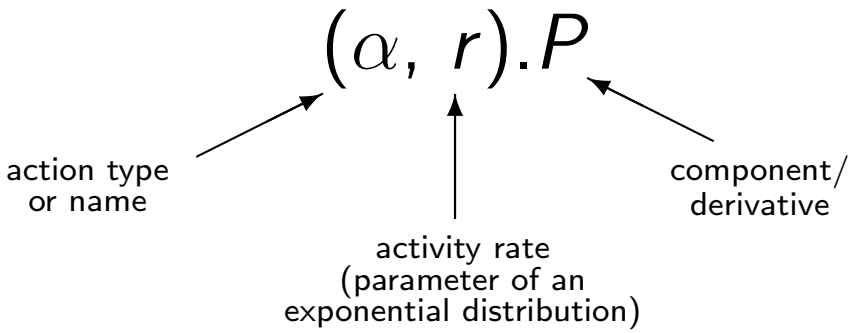
- Closed form analytical models (queueing networks);
- Discrete event simulations; or
- Numerical solution of continuous time Markov chains (CTMC) (Stochastic Petri nets, Stochastic process algebras etc.)

Each of these models is based on a discrete state space and continuous time.

The major limitations of the SPA/CTMC approach are the state space explosion problem and the reliance on exponential distributions.

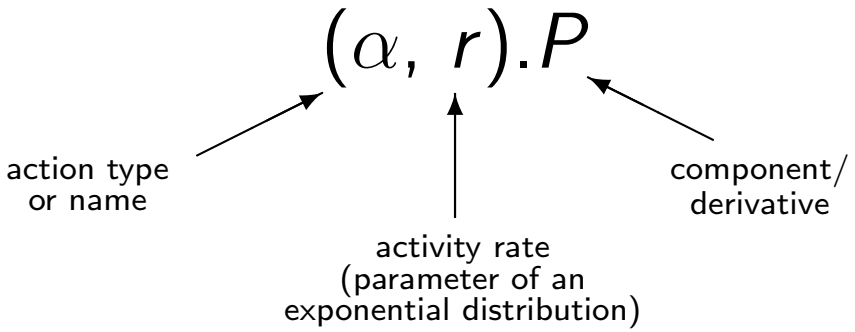
Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.



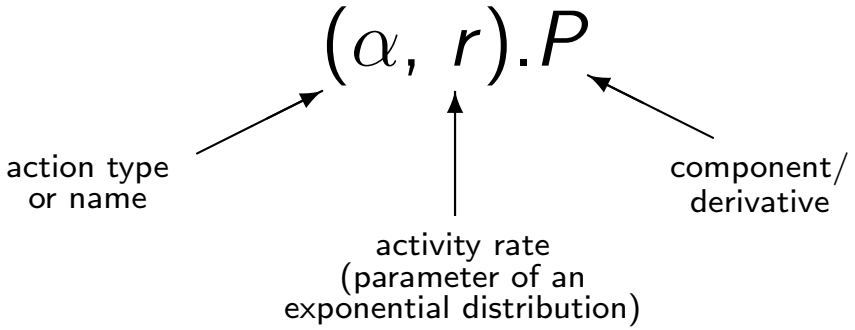
Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.



Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.

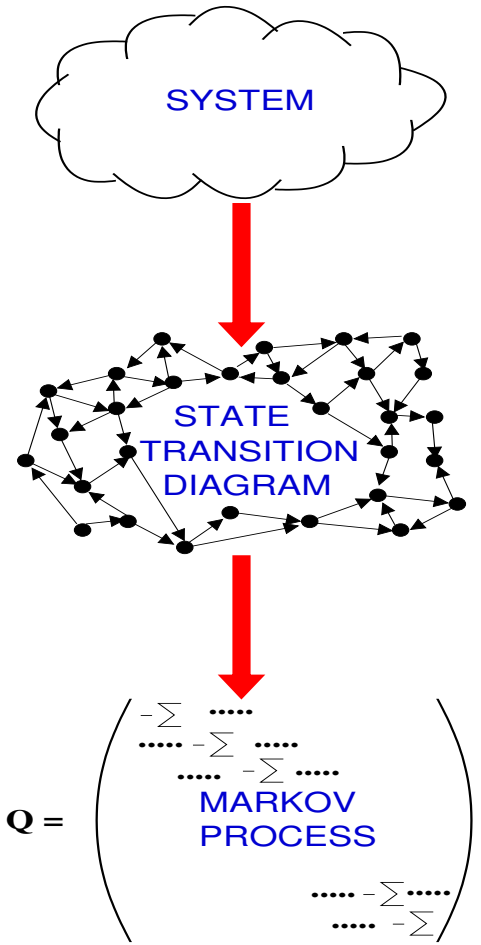


In a PEPA model the state at any current time is the local derivative or state of each component of the model.

Performance Modelling using CTMC

Model Construction

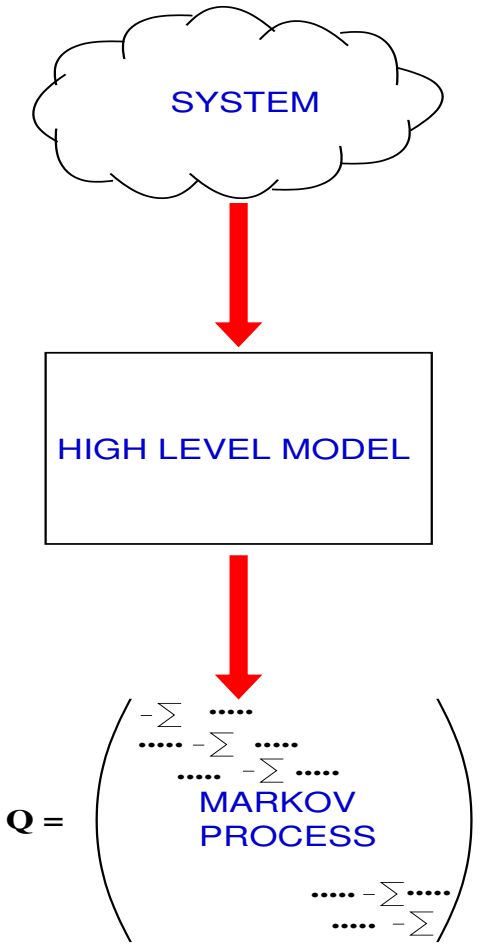
- describing the system using a high level modelling formalism
- generating the underlying CTMC



Performance Modelling using CTMC

Model Construction

- describing the system using a high level modelling formalism
- generating the underlying CTMC



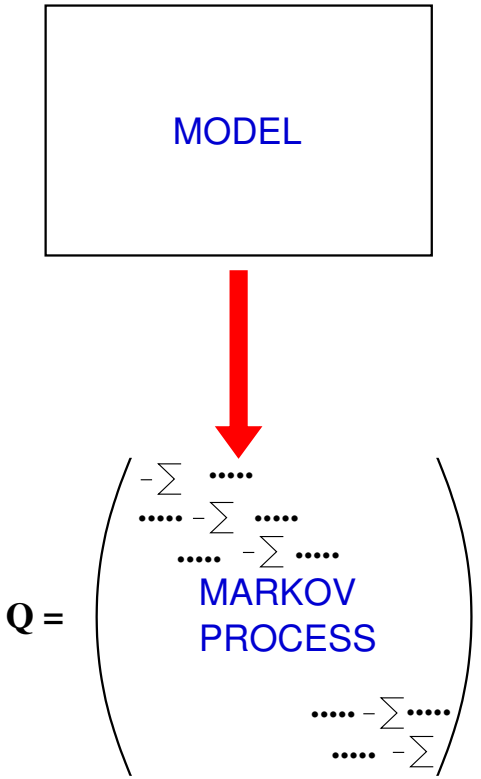
Performance Modelling using CTMC

Model Construction

- describing the system using a high level modelling formalism
- generating the underlying CTMC

Model Manipulation

- model simplification
- model aggregation



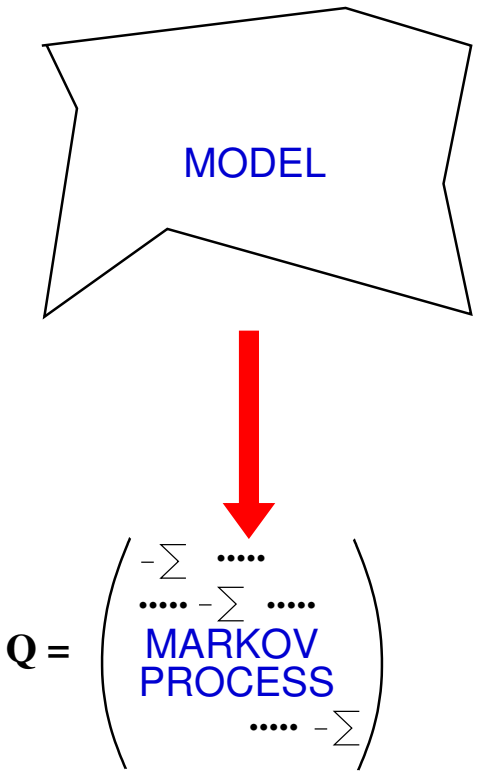
Performance Modelling using CTMC

Model Construction

- describing the system using a high level modelling formalism
- generating the underlying CTMC

Model Manipulation

- model simplification
- model aggregation



Performance Modelling using CTMC

Model Construction

- describing the system using a high level modelling formalism
- generating the underlying CTMC

Model Manipulation

- model simplification
- model aggregation

Model Solution

- solving the CTMC to find steady state probability distribution
- deriving performance measures

Model Manipulation

Model simplification: use a model-model equivalence to substitute one model by another which is more attractive from a solution point of view, e.g. smaller state space, special class of model, etc.

Model Manipulation

Model simplification: use a model-model equivalence to substitute one model by another which is more attractive from a solution point of view, e.g. smaller state space, special class of model, etc.

Model aggregation: use a state-state equivalence to establish a partition of the state space of a model, and replace each set of states by one macro-state, i.e. take a different stochastic representation of the same model.

Outline

Aggregation and lumpability

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

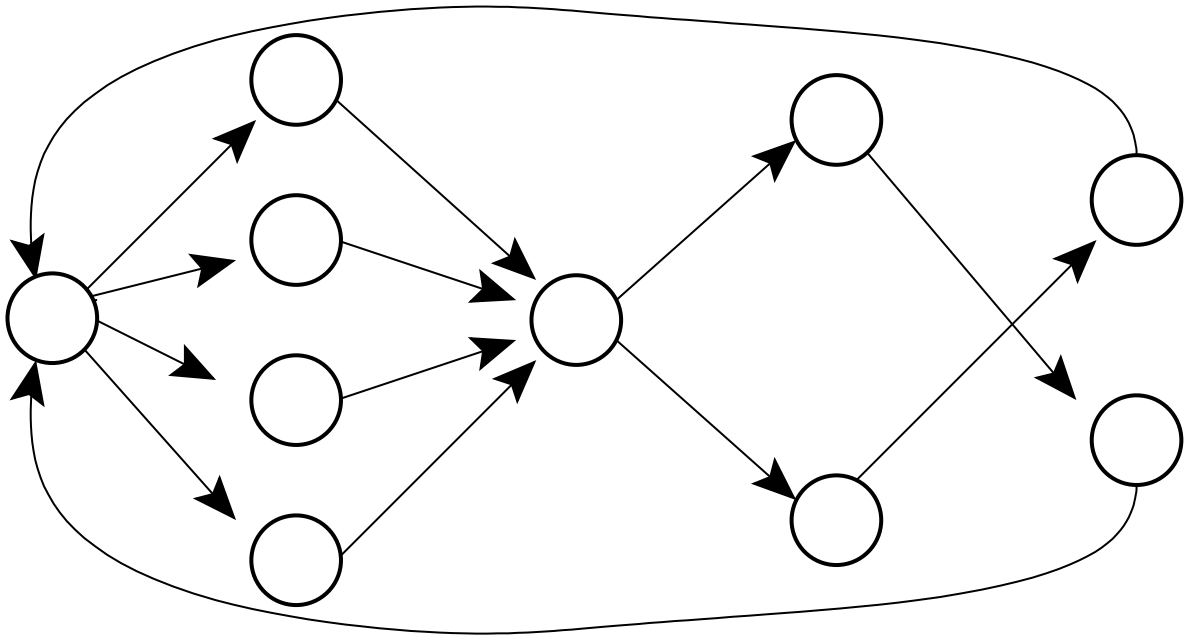
Aggregation and lumpability

- Model aggregation: use a state-state equivalence to establish a partition of 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.

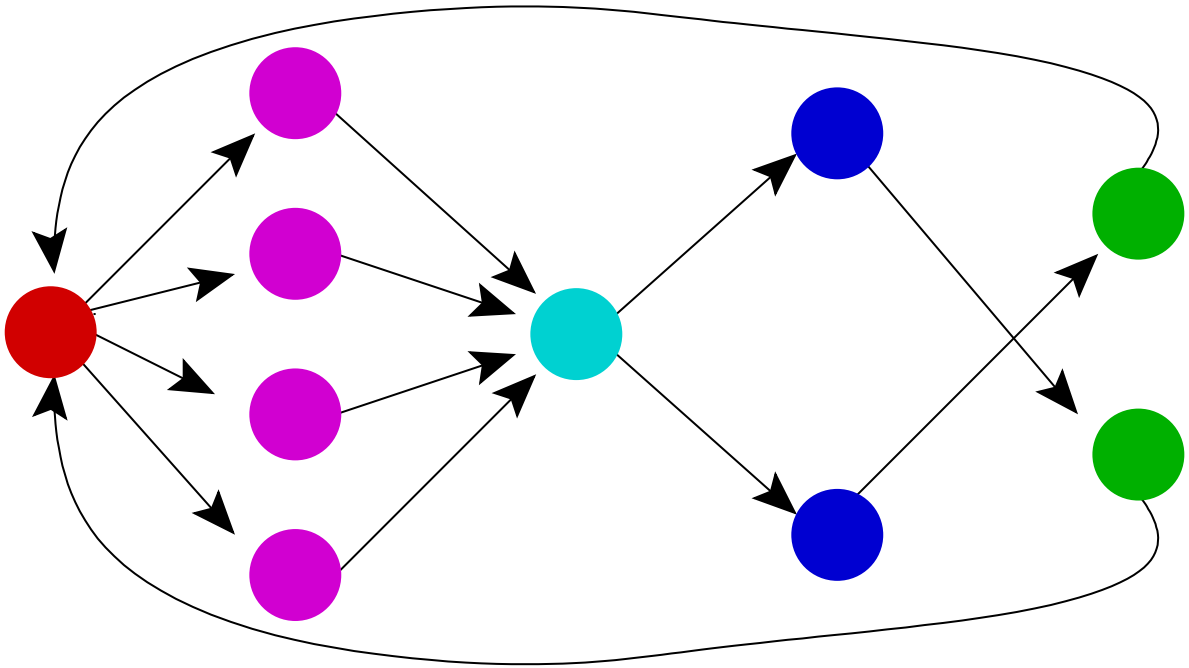
Aggregation and lumpability

- Model aggregation: use a state-state equivalence to establish a partition of 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.
- A lumpable partition is the only partition of a Markov process which preserves the Markov property.

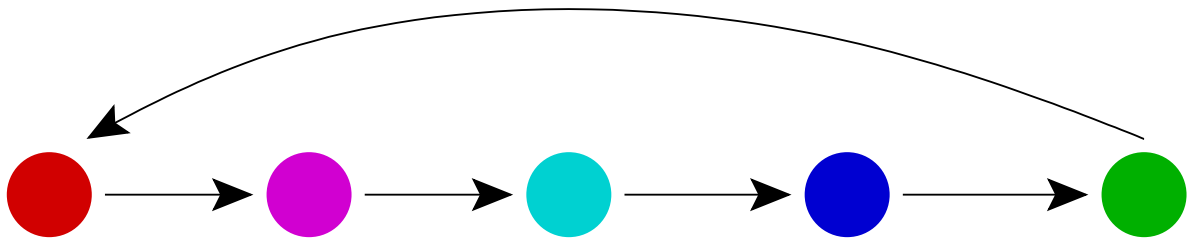
Reducing by lumpability



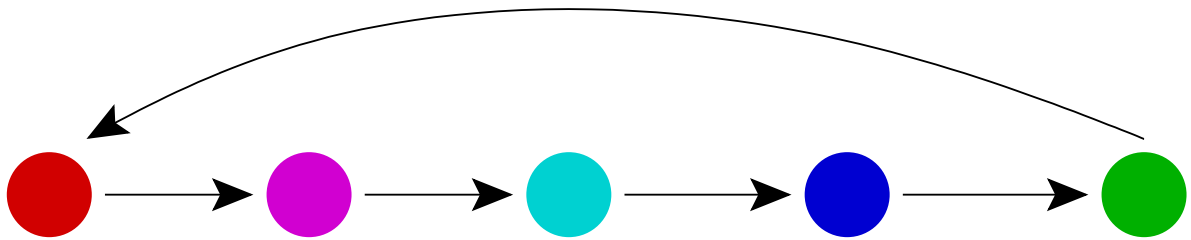
Reducing by lumpability



Reducing by lumpability



Reducing by lumpability



As appealling as this is, it is not the case that it is always mathematically legitimate.

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

Lumpability

- In the early 1960's Kemeny and Snell established the conditions under which it was possible to lump a Markov chain and still have a Markov chain afterwards.

Lumpability

- In the early 1960's Kemeny and Snell established the conditions under which it was possible to lump a Markov chain and still have a Markov chain afterwards.
- In particular these conditions were characterised by conditions on the rates which are straightforward to check.

Lumpability

- In the early 1960's Kemeny and Snell established the conditions under which it was possible to lump a Markov chain and still have a Markov chain afterwards.
- In particular these conditions were characterised by conditions on the rates which are straightforward to check.
- However checking the conditions did involve constructing the complete Markov chain first.

Lumpability

- In the early 1960's Kemeny and Snell established the conditions under which it was possible to lump a Markov chain and still have a Markov chain afterwards.
- In particular these conditions were characterised by conditions on the rates which are straightforward to check.
- However checking the conditions did involve constructing the complete Markov chain first.
- This is something of a catch-22 situation when the problem is that the state space of the Markov chain is too large to handle.

Lumpability

If the original state space is $\{X_1, X_2, \ldots, X_n\}$ then the aggregated state space is some $\{X_{[1]}, X_{[2]}, \ldots, X_{[N]}\}$ where $N < n$ and ideally $N << n$.

Lumpability

If the original state space is $\{X_1, X_2, \ldots, X_n\}$ then the aggregated state space is some $\{X_{[1]}, X_{[2]}, \ldots, X_{[N]}\}$ where $N < n$ and ideally $N << n$.

If the transition rates of the original process are $q(X_i, X_k)$ then the transition rates into any partition from a state is

$$q(X_i, X_{[j]}) = \sum_{k \in [j]} q(X_i, X_j)$$

Lumpability

If the original state space is $\{X_1, X_2, \ldots, X_n\}$ then the aggregated state space is some $\{X_{[1]}, X_{[2]}, \ldots, X_{[N]}\}$ where $N < n$ and ideally $N << n$.

If the transition rates of the original process are $q(X_i, X_k)$ then the transition rates into any partition from a state is

$$q(X_i, X_{[j]}) = \sum_{k \in [j]} q(X_i, X_j)$$

Transition rates between partitions are the weighted sum of the transition rates of each state in the first partition to the second partition, weighted by the conditional steady state probability of that state in the partition, $\overline{\pi}_j(\cdot)$

$$q(X_{[j]}, X_{[i]}) = \sum_{k \in [j]} \overline{\pi}_j(X_k) q(X_k, X_{[i]})$$

Ordinary, Exact and Strict Lumpability

- A Markov process is ordinarily lumpable with respect to a partition $\chi = \{X_{[i]}\}$ iff, for any $X_{[k]}, X_{[l]} \in \chi, X_i, X_j \in X_{[k]}$

$$q(X_i, X_{[l]}) = q(X_j, X_{[l]})$$

Ordinary, Exact and Strict Lumpability

- A Markov process is ordinarily lumpable with respect to a partition $\chi = \{X_{[i]}\}$ iff, for any $X_{[k]}, X_{[l]} \in \chi, X_i, X_j \in X_{[k]}$

$$q(X_i, X_{[l]}) = q(X_j, X_{[l]})$$

- χ is an exactly lumpable partition iff, for any $X_{[k]}, X_{[l]} \in \chi, X_i, X_j \in X_{[l]}$

$$q(X_{[k]}, X_i) = q(X_{[k]}, X_j)$$

Ordinary, Exact and Strict Lumpability

- A Markov process is ordinarily lumpable with respect to a partition $\chi = \{X_{[i]}\}$ iff, for any $X_{[k]}, X_{[l]} \in \chi, X_i, X_j \in X_{[k]}$

$$q(X_i, X_{[l]}) = q(X_j, X_{[l]})$$

- χ is an exactly lumpable partition iff, for any $X_{[k]}, X_{[l]} \in \chi, X_i, X_j \in X_{[l]}$

$$q(X_{[k]}, X_i) = q(X_{[k]}, X_j)$$

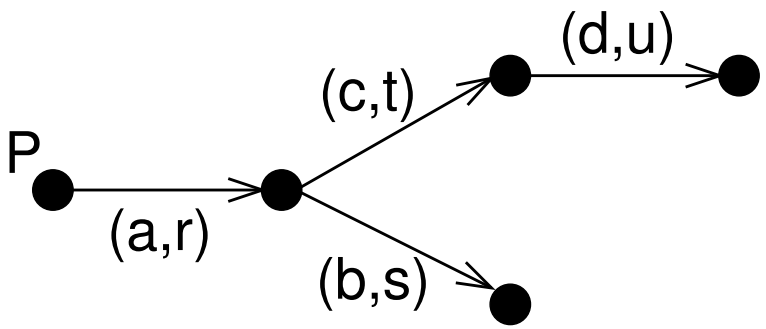
- χ is a strictly lumpable partition iff it is ordinarily lumpable and exactly lumpable.

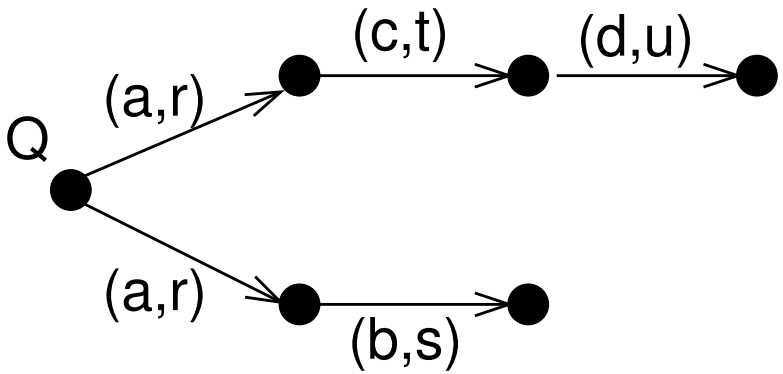
Strong Equivalence in PEPA

Strong equivalence in PEPA is a bisimulation in the style of Larsen of Skou.

Strong Equivalence in PEPA

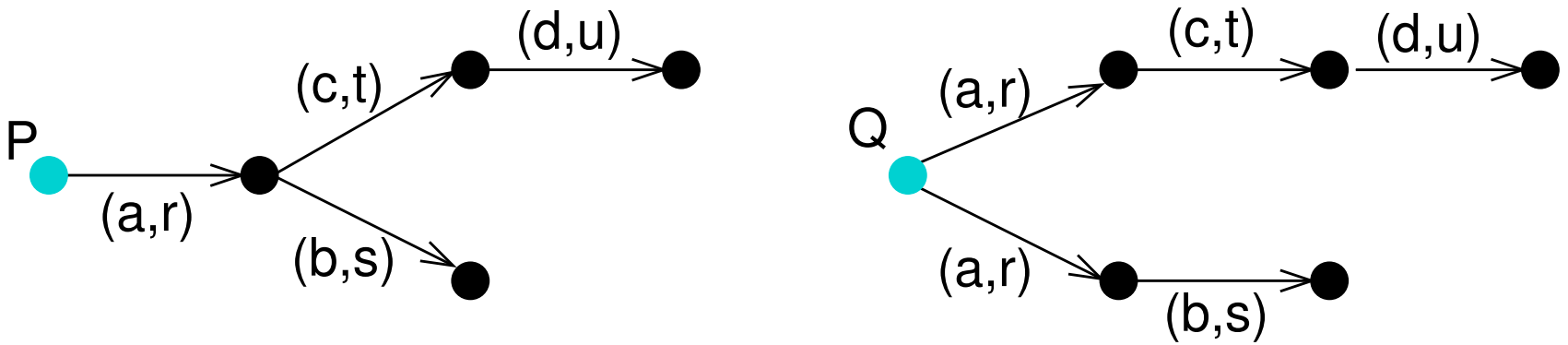
Strong equivalence in PEPA is a bisimulation in the style of Larsen of Skou.





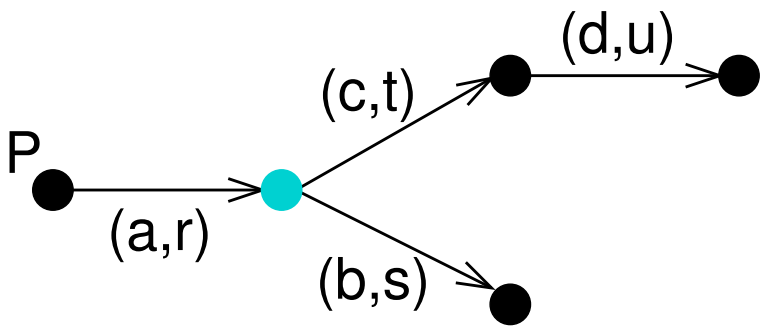
Strong Equivalence in PEPA

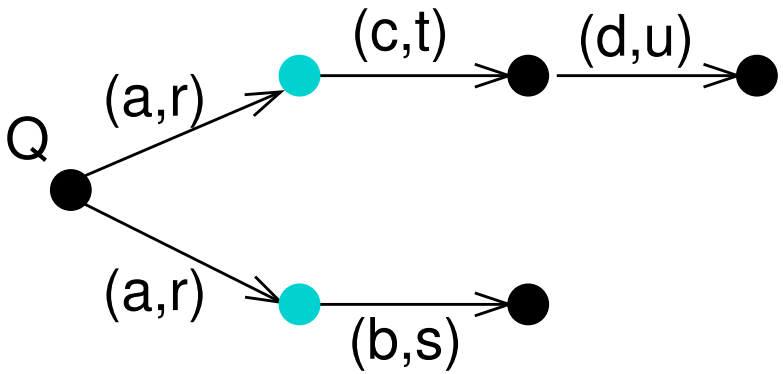
Strong equivalence in PEPA is a bisimulation in the style of Larsen of Skou.



Strong Equivalence in PEPA

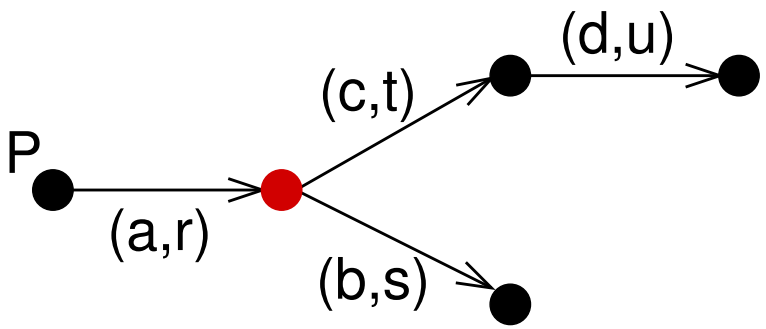
Strong equivalence in PEPA is a bisimulation in the style of Larsen of Skou.

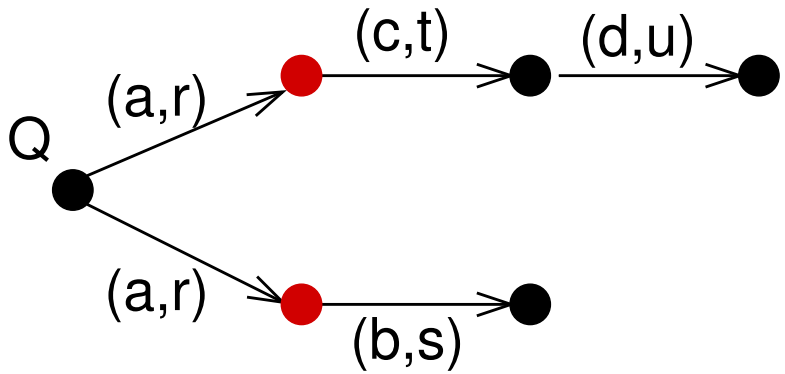




Strong Equivalence in PEPA

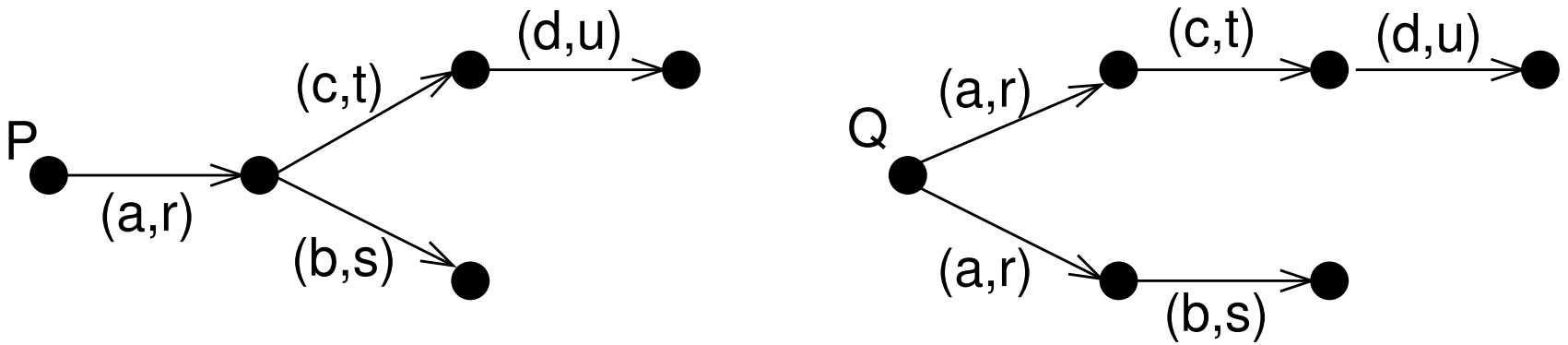
Strong equivalence in PEPA is a bisimulation in the style of Larsen of Skou.





Strong Equivalence in PEPA

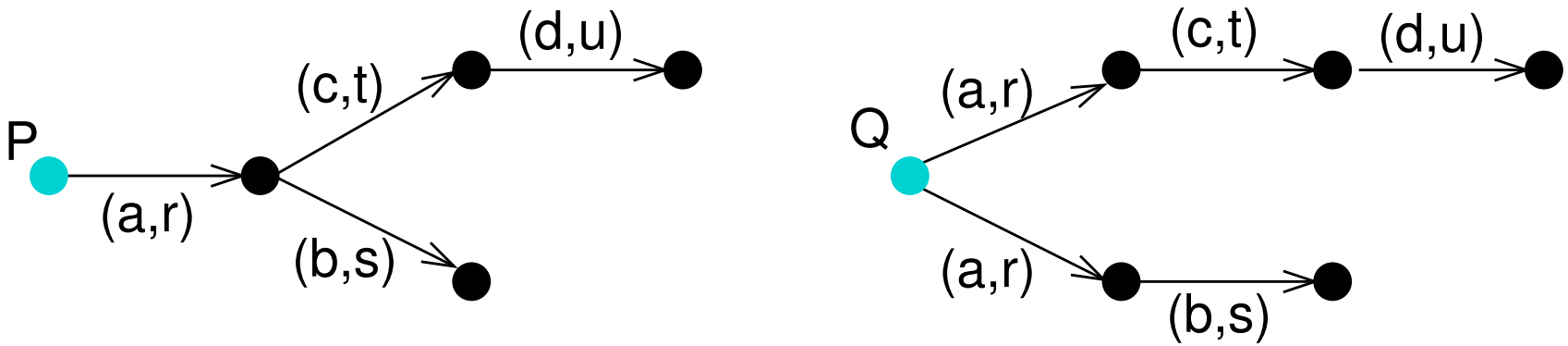
Strong equivalence in PEPA is a bisimulation in the style of Larsen of Skou.



Observability is assumed to include the ability to record timing information over a number of runs.

Strong Equivalence in PEPA

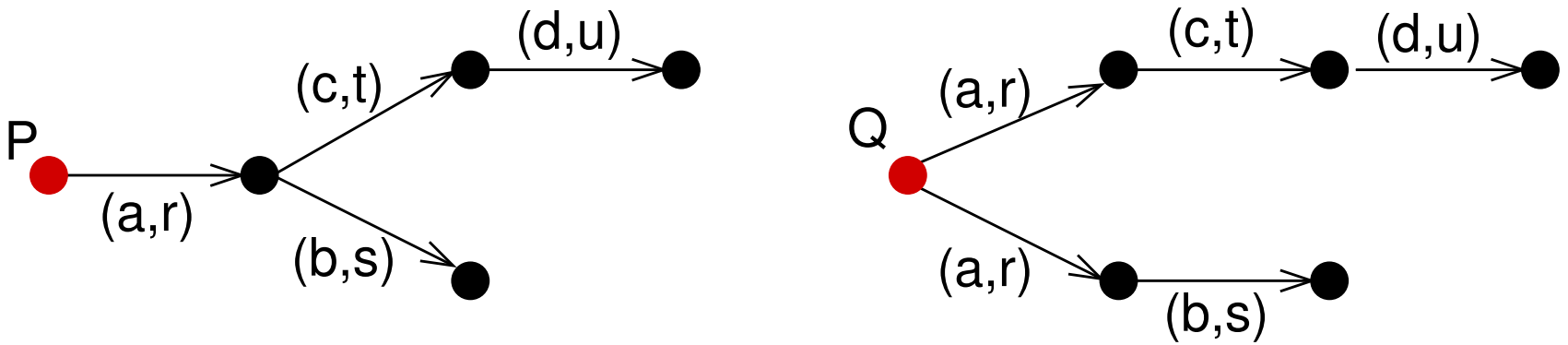
Strong equivalence in PEPA is a bisimulation in the style of Larsen of Skou.



Observability is assumed to include the ability to record timing information over a number of runs.

Strong Equivalence in PEPA

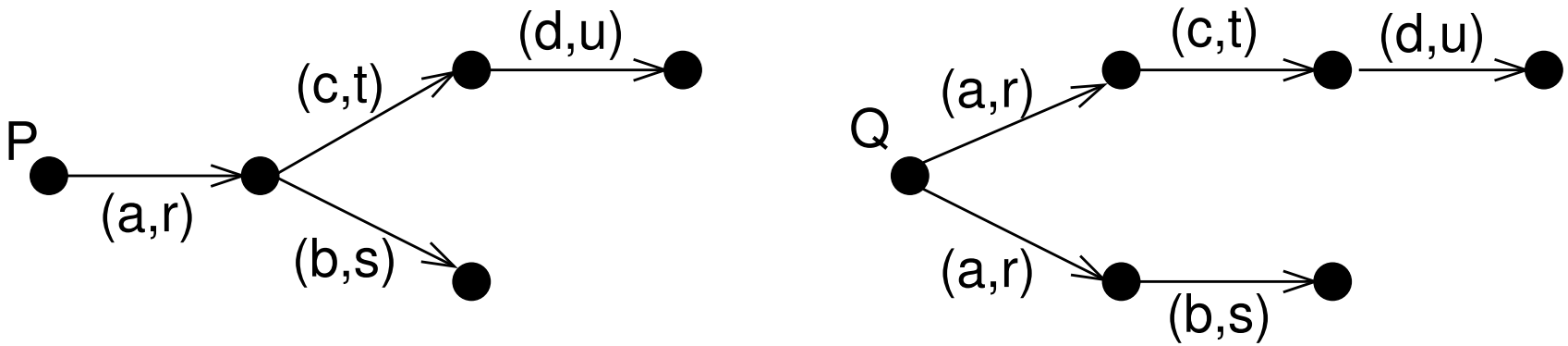
Strong equivalence in PEPA is a bisimulation in the style of Larsen of Skou.



Observability is assumed to include the ability to record timing information over a number of runs.

Strong Equivalence in PEPA

Strong equivalence in PEPA is a bisimulation in the style of Larsen of Skou.

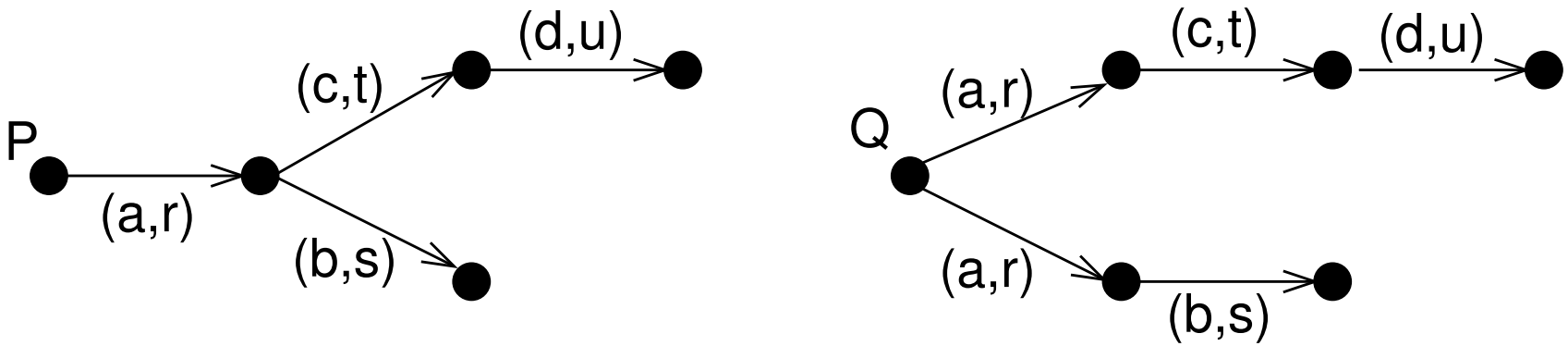


Observability is assumed to include the ability to record timing information over a number of runs.

Two processes are equivalent if they can undertake the same actions, at the same rate, and arrive at processes that are equivalent.

Strong Equivalence in PEPA

Strong equivalence in PEPA is a bisimulation in the style of Larsen of Skou.



Observability is assumed to include the ability to record timing information over a number of runs.

Two processes are equivalent if they can undertake the same actions, at the same rate, and arrive at processes that are equivalent.

Expressed as rates to equivalence classes of processes

Strong Equivalence in PEPA

Definition

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

$$q[P, S, \alpha] = q[Q, S, \alpha].$$

where

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

Strong Equivalence and Lumpability

- Given this definition it is fairly straightforward to show that if we consider strong equivalence of states within a single model, it induces an ordinarily lumpable partition on the state space of the underlying Markov chain.

Strong Equivalence and Lumpability

- Given this definition it is fairly straightforward to show that if we consider strong equivalence of states within a single model, it induces an ordinarily lumpable partition on the state space of the underlying Markov chain.
- Moreover it can be shown that strong equivalence is a congruence.

Strong Equivalence and Lumpability

- Given this definition it is fairly straightforward to show that if we consider strong equivalence of states within a single model, it induces an ordinarily lumpable partition on the state space of the underlying Markov chain.
- Moreover it can be shown that strong equivalence is a congruence.
- This means that aggregation based on lumpability can be applied component by component, avoiding the previous problem of having to construct the complete state space in order to find the lumpable partitions.

Simple Example PEPA Model

PEPA Model

$$User_1 \stackrel{\text{def}}{=} (task_1, 1).User_2$$

$$User_2 \stackrel{\text{def}}{=} (task_2, 1).User_1$$

$$Provider_1 \stackrel{\text{def}}{=} (task_1, 1).Provider_2$$

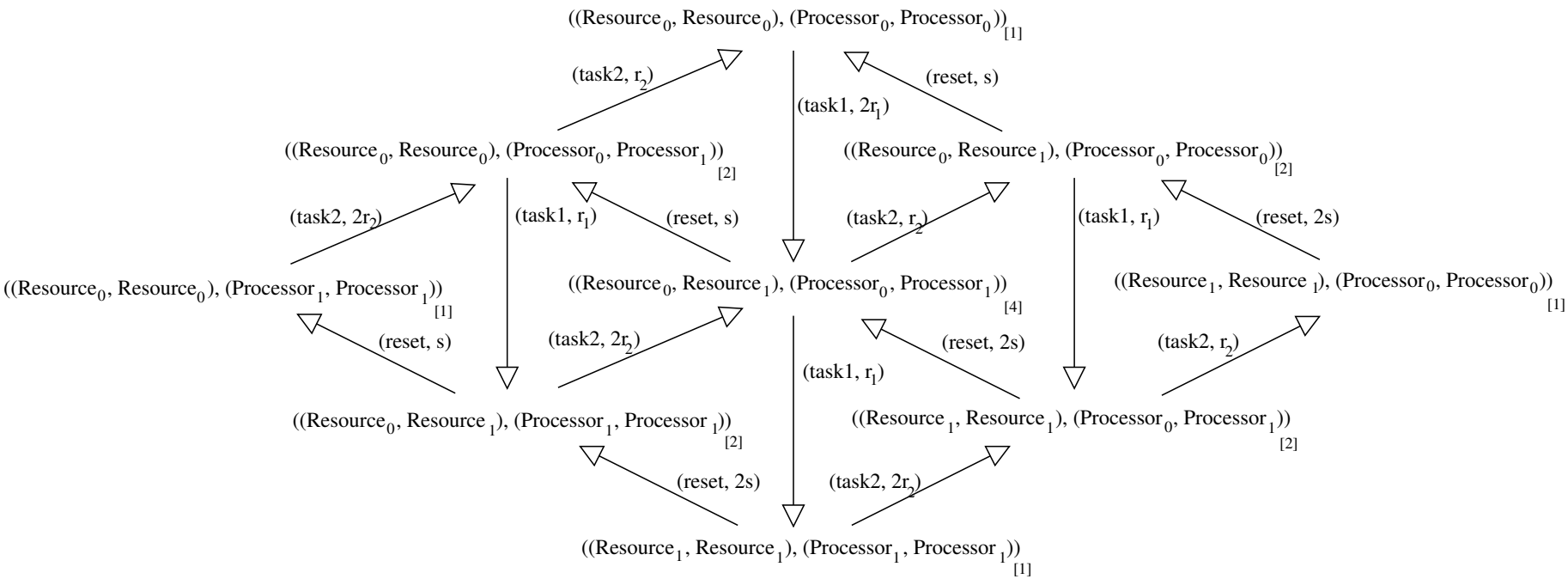
$$Provider_2 \stackrel{\text{def}}{=} (reset, 1).Provider_1$$

$$\underbrace{User_1 || \cdots || User_1}_{M \text{ copies}} \underset{\{task1\}}{\bowtie} \underbrace{Provider_1 || \cdots || Provider_1}_{N \text{ copies}}$$

The size of state space: $2^M \times 2^N$.

Aggregation in PEPA

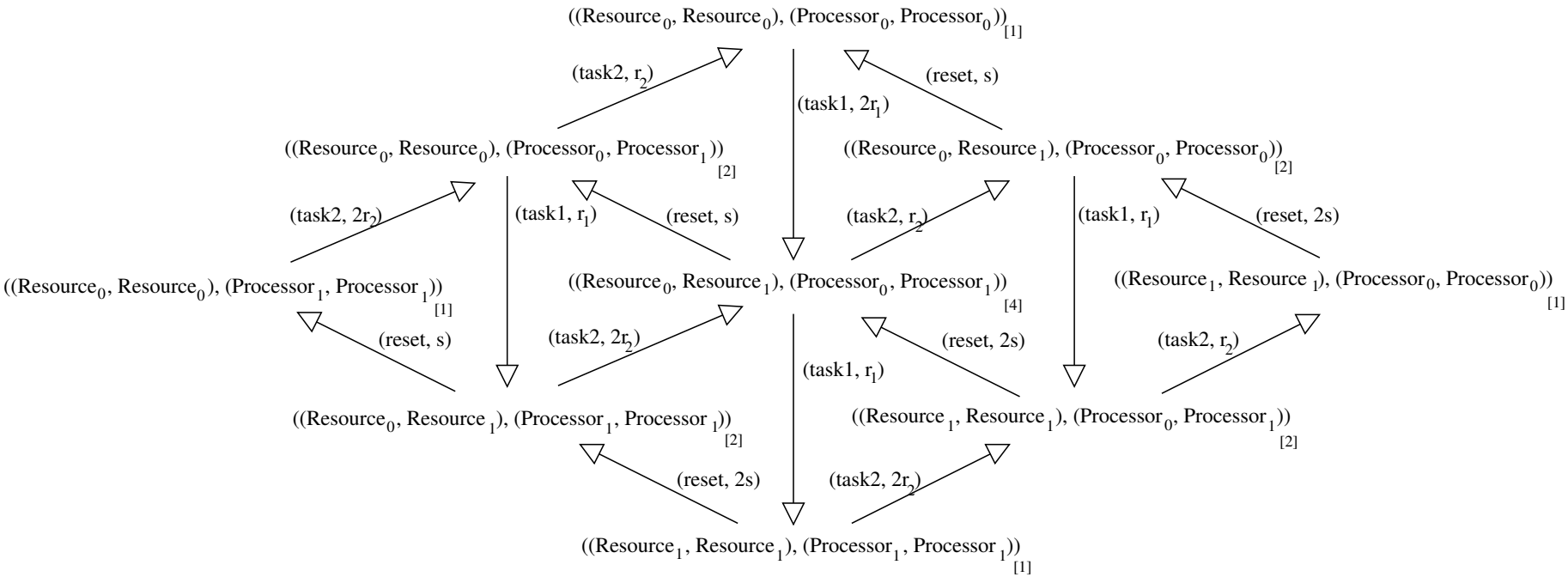
For $N = M = 2$:



a) Aggregated state space in canonical form

Aggregation in PEPA

For $N = M = 2$:



a) Aggregated state space in canonical form

The state space size reduces to $(M + 1) \times (N + 1)$.

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.

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.

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 Ribaudo, IEEE TSE 2001].

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 Ribaudo, IEEE TSE 2001].

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

Achieving aggregration

- 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 $C_i, i = 1, 2, \cdots, n$, 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 $\mathbf{m}[C_{i_j}]$ records how many instances of the jth 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.

Numerical vector form

For our example model:

$$\mathbf{m} = (\mathbf{m}[User_1], \mathbf{m}[User_2], \mathbf{m}[Provider_1], \mathbf{m}[Provider_2])^T .$$

Numerical vector form

For our example model:

$$\mathbf{m} = (\mathbf{m}[User_1], \mathbf{m}[User_2], \mathbf{m}[Provider_1], \mathbf{m}[Provider_2])^T .$$

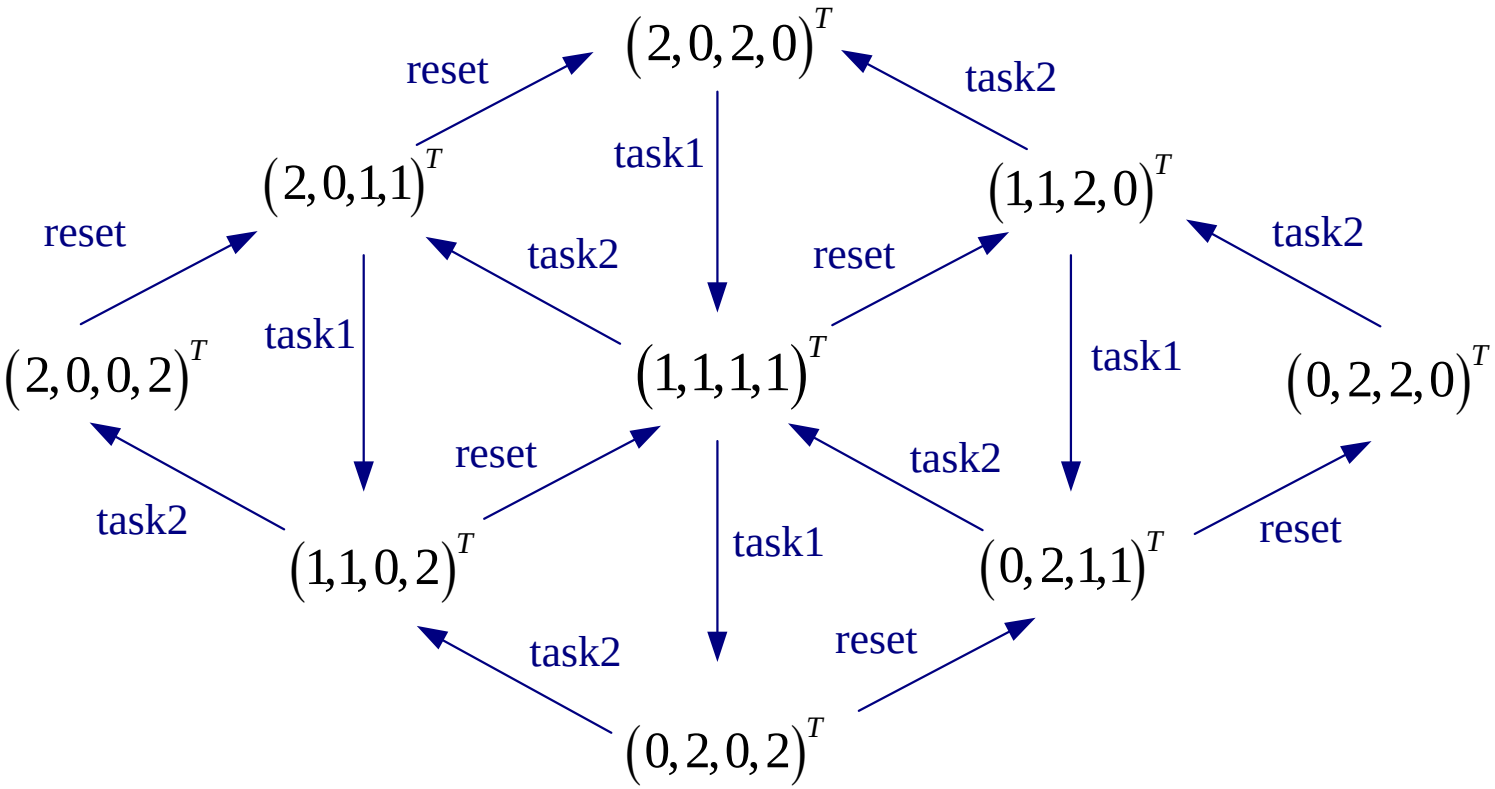
Let $M = N = 2$. The system equation of the model determines the starting state:

$$\mathbf{m} = (M, 0, N, 0)^T = (2, 0, 2, 0)^T$$

Exercising the activities of the model the reachable states are found.

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

The resulting state space



The size of the aggregated state space

Proposition

Consider a system comprising n types of component, namely $C_1, C_2, \cdots, C_n$, with m_i copies of the component of type C_i in the system, where C_i has d_i local derivatives, for $i = 1, 2, \cdots, n$. Then the size of the state space of the system is at most

$$\prod_{i=1}^{n} (m_i + d_i - 1)^{d_i - 1}.$$

Efficiency: the size of the state space increase at most polynomially with the number of components, by adopting numerical vector forms.

In the example model...

In our model, *User* has two local states: $User_1$ and $User_2$; *Provider* has two local states: $Provider_1$ and $Provider_2$. An upper bound of the size given by the proposition is

$$(M + 2 - 1)^{2-1}(N + 2 - 1)^{2-1} = (M + 1) \times (N + 1).$$

In the example model...

In our model, *User* has two local states: $User_1$ and $User_2$; *Provider* has two local states: $Provider_1$ and $Provider_2$. An upper bound of the size given by the proposition is

$$(M + 2 - 1)^{2-1}(N + 2 - 1)^{2-1} = (M + 1) \times (N + 1).$$

Let $M = N = 2$, the upper bound is $(2 + 1) \times (2 + 1) = 9$, coinciding with the size of the state space.

In the example model...

In our model, *User* has two local states: $User_1$ and $User_2$; *Provider* has two local states: $Provider_1$ and $Provider_2$. An upper bound of the size given by the proposition is

$$(M + 2 - 1)^{2-1}(N + 2 - 1)^{2-1} = (M + 1) \times (N + 1).$$

Let $M = N = 2$, the upper bound is $(2 + 1) \times (2 + 1) = 9$, coinciding with the size of the state space.

It is a significant improvement to reduce the size of the state space from $2^M \times 2^N$ to $(M + 1) \times (N + 1)$, without relevant information and accuracy loss.

Activity matrix

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

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.

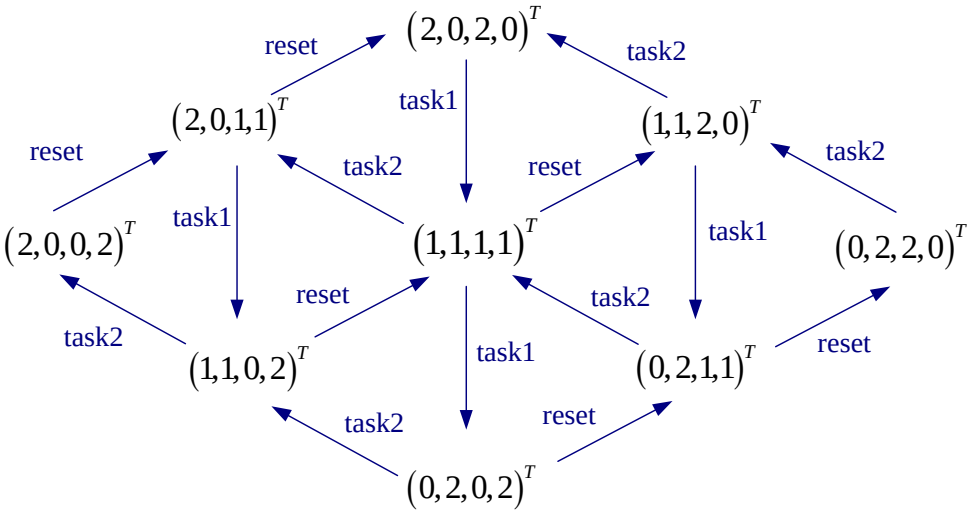
Activity matrix

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

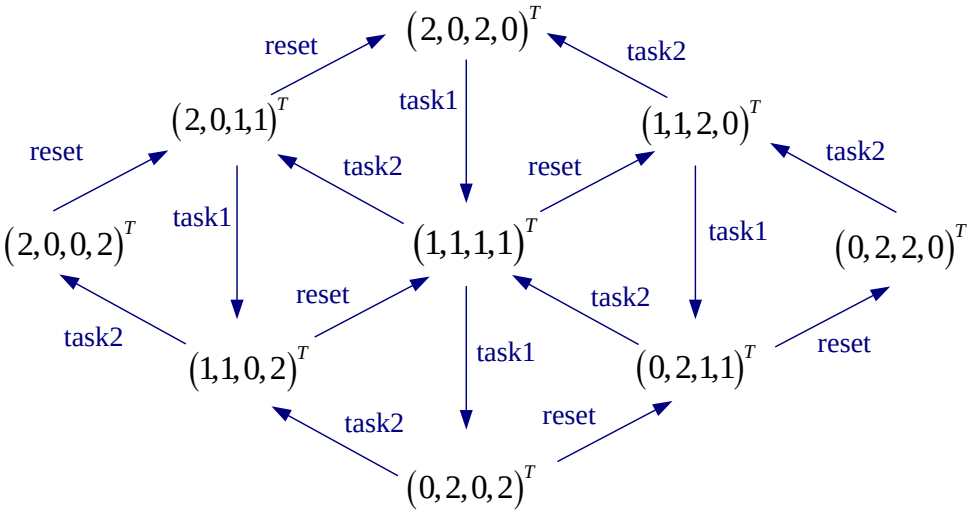
The transitions arise from the activities in the PEPA model. Thus we use the syntax of the process algebra to infer the impact of each activity on the current state, numerically.

This information is represented in the activity matrix.

Activity matrix

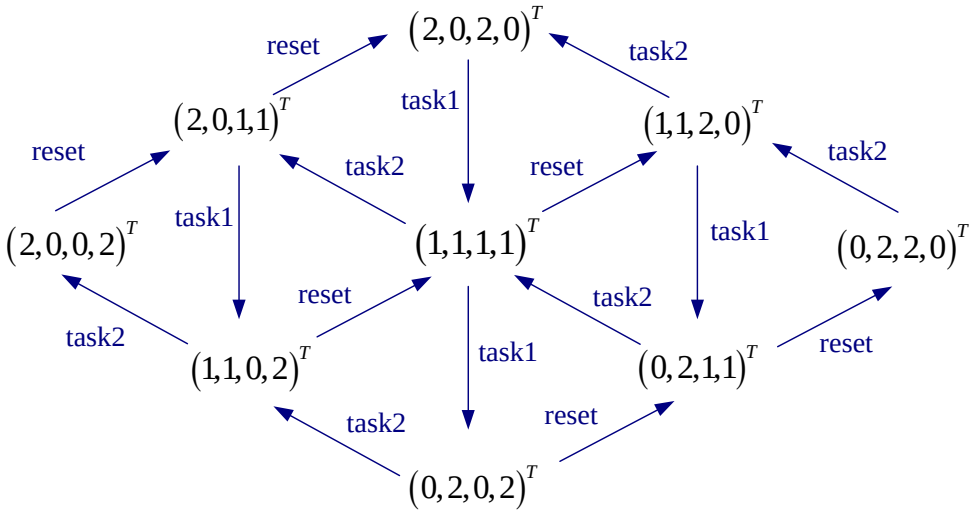


Activity matrix



	I^{task_1}	I^{task_2}	I^{reset}
U_1	-1	1	0
U_2	1	-1	0
P_1	-1	0	1
P_2	1	0	-1

Activity matrix



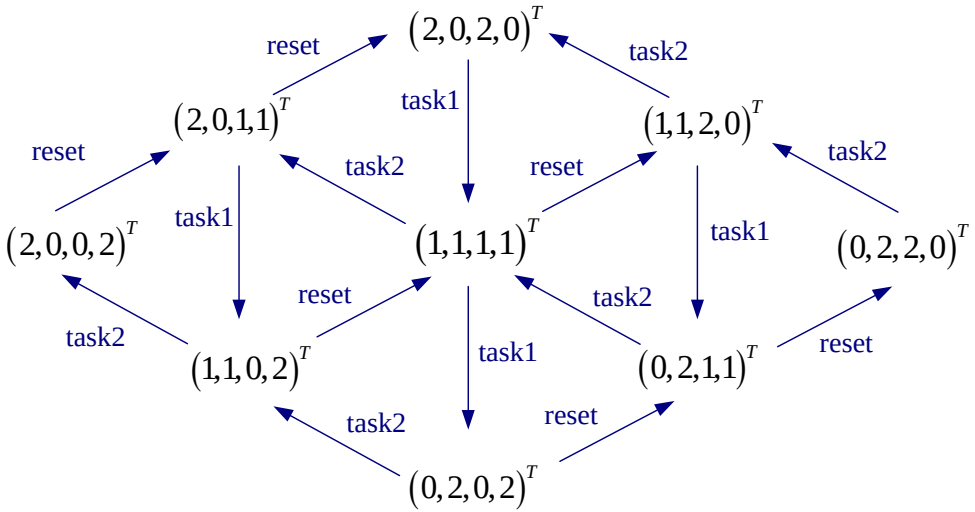
	I^{task_1}	I^{task_2}	I^{reset}
U_1	-1	1	0
U_2	1	-1	0
P_1	-1	0	1
P_2	1	0	-1

Each activity corresponds a transition vector. All transition vectors form an *activity matrix* **C**. For example,

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

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

Activity matrix



	I^{task_1}	I^{task_2}	I^{reset}
U_1	-1	1	0
U_2	1	-1	0
P_1	-1	0	1
P_2	1	0	-1

Each activity corresponds a transition vector. All transition vectors form an *activity matrix* **C**. For example,

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

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

Moreover, any state can be represented by C and starting state:
$\mathbf{m} = \mathbf{m}_0 + \mathbf{C}\boldsymbol{\sigma} = (2,0,2,0)^T + \mathbf{C}\boldsymbol{\sigma}$.

Model 2

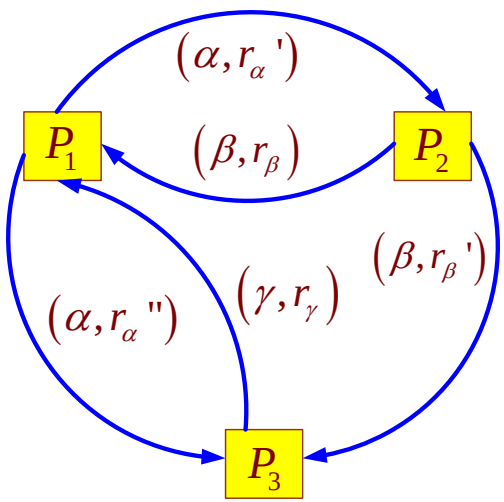
PEPA Model

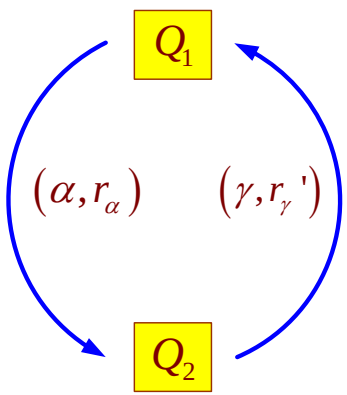
$$P_1 \stackrel{\text{def}}{=} (\alpha, r'_\alpha).P_2 + (\alpha, r''_\alpha).P_3 \qquad Q_1 \stackrel{\text{def}}{=} (\alpha, r_\alpha).Q_2$$

$$P_2 \stackrel{\text{def}}{=} (\beta, r_\beta).P_1 + (\beta, r'_\beta).P_3 \qquad Q_2 \stackrel{\text{def}}{=} (\gamma, r'_\gamma).Q_1$$

$$P_3 \stackrel{\text{def}}{=} (\gamma, r_\gamma).P_1$$

$$P_1[A] \underset{\{\alpha\}}{\bowtie} Q_1[B].$$





Activity matrix of Model 2

	α	β	γ
P_1	-1	1	1
P_2	1	-1	0
P_3	1	1	-1
Q_1	-1	0	1
Q_2	1	0	-1

The original activity matrix cannot reflect:

- Which activity is shared, which is individual;
- After firing an activity, only one output will be chosen;
- The correspondence between activities and state transitions.

Labelled activity matrix

We attach labels (represented by "pre" and "post" local derivatives) to distinguish these cases....

	$\alpha^{(P_1 \to P_2, Q_1 \to Q_2)}$	$\alpha^{(P_1 \to P_3, Q_1 \to Q_2)}$	$\beta^{P_2 \to P_1}$	$\beta^{P_2 \to P_3}$	$\gamma^{P_3 \to P_1}$	$\gamma^{Q_2 \to Q_1}$
P_1	-1	-1	1	0	1	0
P_2	1	0	-1	-1	0	0
P_3	0	1	0	1	-1	0
Q_1	-1	-1	0	0	0	1
Q_2	1	1	0	0	0	-1

Pre and post local derivatives

Definition

1. If a local derivative U can enable an activity l, that is $U \xrightarrow{l} \cdot$, then U is called a *pre local derivative* of l. The set of all pre local derivatives of l is denoted by $\mathrm{pre}(l)$, called the *pre set* of l.
2. If V is a local derivative obtained by firing an activity l, i.e. $\cdot \xrightarrow{l} V$, then V is called a *post local derivative* of l. The set of all post local derivatives is denoted by $\mathrm{post}(l)$, called the *post set* of l.
3. The set of all the local derivatives derived from U by firing l, i.e.

$$\mathrm{post}(U, l) = \{V \mid U \xrightarrow{l} V\},$$

is called the *post set of l from U*.

N.B. We disregard rates for now as they have no impact.

Labelled activity

Definition

1. For any individual activity l, for each $U \in \mathrm{pre}(l), V \in \mathrm{post}(U, l)$, label l as $l^{U \to V}$.
2. For a shared activity l, for each $(V_1, V_2, \cdots, V_k)$ in

$$\mathrm{post}(\mathrm{pre}(l)[1], l) \times \mathrm{post}(\mathrm{pre}(l)[2], l) \times \cdots \times \mathrm{post}(\mathrm{pre}(l)[k], l),$$

label l as l^w, where

$$w = (\mathrm{pre}(l)[1] \to V_1, \mathrm{pre}(l)[2] \to V_2, \cdots, \mathrm{pre}(l)[k] \to V_k).$$

Each $l^{U \to V}$ or l^w is called a *labelled activity*. The set of all labelled activities is denoted by $\mathcal{A}_{\mathrm{label}}$.

Idea: Each pair of pre and corresponding post local derivatives is taken as a label.

Labelled activity matrix

Definition

For a model with $N_{\mathcal{A}_{\text{label}}}$ labelled activities and $N_{\mathcal{D}}$ distinct local derivatives, the activity matrix $\mathbf{C}$ is an $N_{\mathcal{D}} \times N_{\mathcal{A}_{\text{label}}}$ matrix, and the entries are defined as follows

$$\mathbf{C}(U_i, l_j) = \begin{cases} +1 & \text{if } U_i \in \text{post}(l_j) \\ -1 & \text{if } U_i \in \text{pre}(l_j) \\ 0 & \text{otherwise} \end{cases}$$

where l_j is a labelled activity.

Correspondence with transitions in the CTMC

Proposition

The mapping between labelled activities (or columns of the activity matrix) and system transitions are one-to-one, i.e., each column of the activity matrix corresponds to a unique system transition and each transition can be represented by a unique column of the activity matrix.

Correspondence with transitions in the CTMC

Proposition

The mapping between labelled activities (or columns of the activity matrix) and system transitions are one-to-one, i.e., each column of the activity matrix corresponds to a unique system transition and each transition can be represented by a unique column of the activity matrix.

Each column of the activity matrix indicates a pair of pre and post local derivatives for each involved component, and therefore represents a single transition in the underlying state space.

Labelled activity matrix of Model 2

	$\alpha^{(P_1 \to P_2, Q_1 \to Q_2)}$	$\alpha^{(P_1 \to P_3, Q_1 \to Q_2)}$	$\beta^{P_2 \to P_1}$	$\beta^{P_2 \to P_3}$	$\gamma^{P_3 \to P_1}$	$\gamma^{Q_2 \to Q_1}$
P_1	-1	-1	1	0	1	0
P_2	1	0	-1	-1	0	0
P_3	0	1	0	1	-1	0
Q_1	-1	-1	0	0	0	1
Q_2	1	1	0	0	0	-1

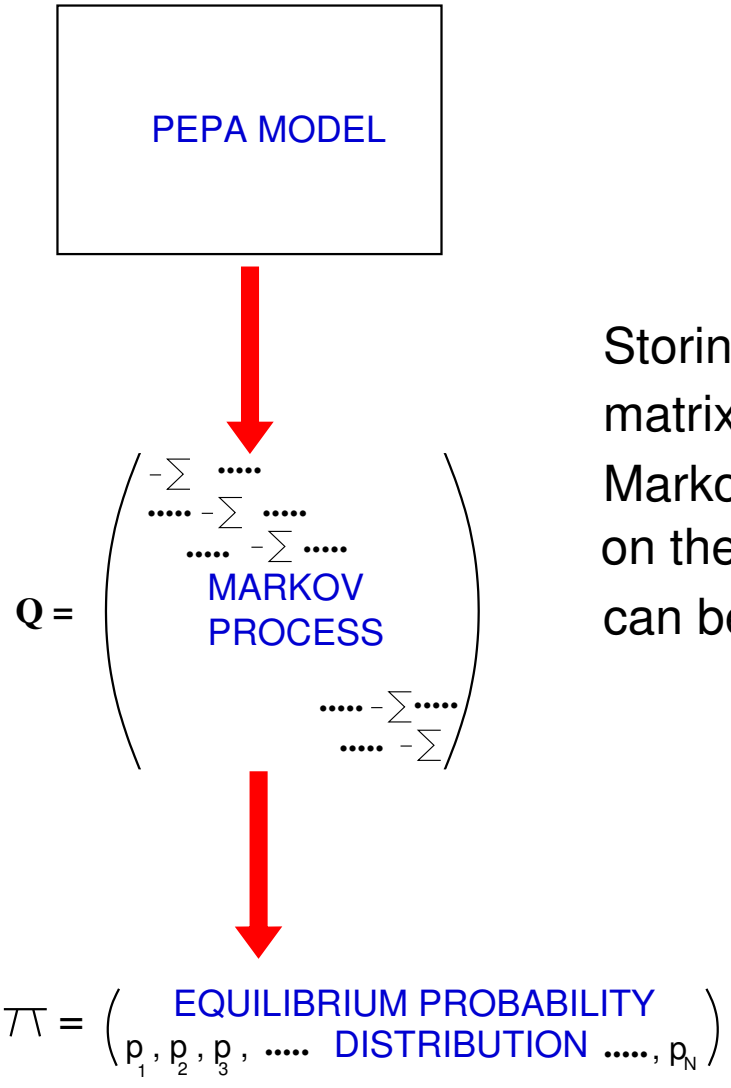
Labelled activity matrix of Model 2

	$\alpha^{(P_1 \to P_2, Q_1 \to Q_2)}$	$\alpha^{(P_1 \to P_3, Q_1 \to Q_2)}$	$\beta^{P_2 \to P_1}$	$\beta^{P_2 \to P_3}$	$\gamma^{P_3 \to P_1}$	$\gamma^{Q_2 \to Q_1}$
P_1	-1	-1	1	0	1	0
P_2	1	0	-1	-1	0	0
P_3	0	1	0	1	-1	0
Q_1	-1	-1	0	0	0	1
Q_2	1	1	0	0	0	-1

All structural information has been represented by labelled activities or (modified) activity matrix.

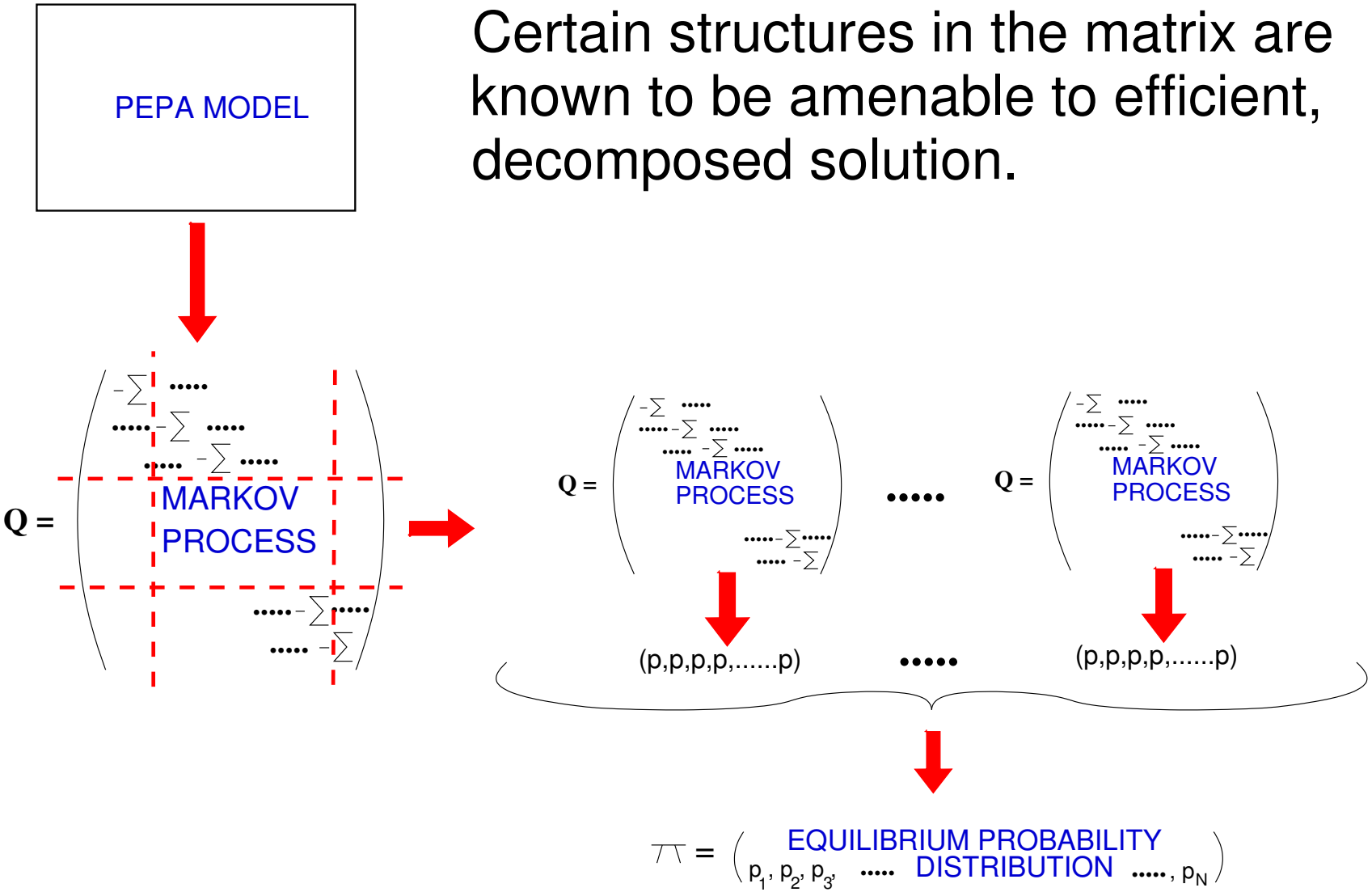
Outline

Characterising efficient solution



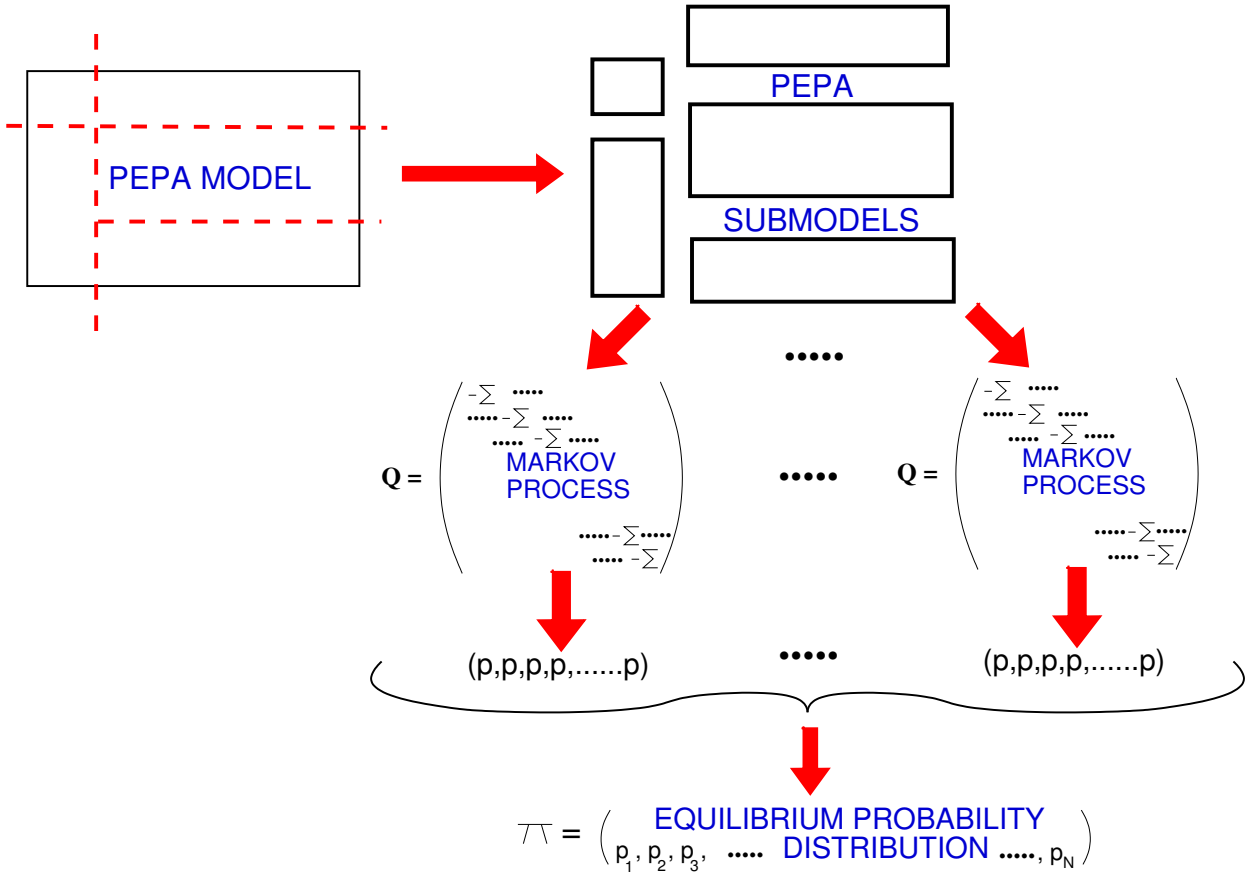
Storing and manipulating the matrix which represents the Markov process places limitations on the size of model which can be analysed.

Characterising efficient solution



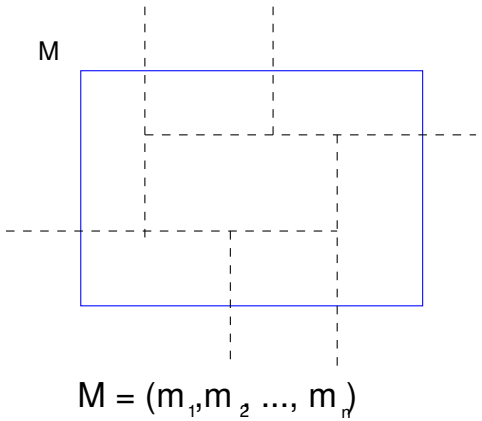
Certain structures in the matrix are known to be amenable to efficient, decomposed solution.

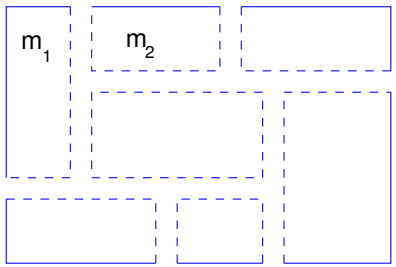
Characterising efficient solution



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





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

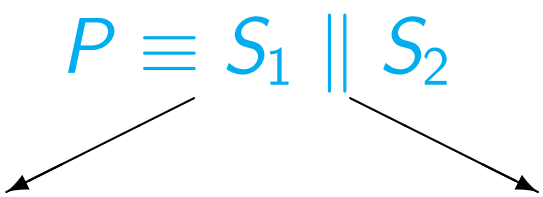
Partition the model M into n statistically independent submodels $m_1, m_2, \ldots, m_n$

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

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



Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted direct interaction between components with a particular structure

$$P \equiv S_1 \underset{L}{\bowtie} S_2$$

S_1, S_2 and L all restricted

Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted direct interaction between components with a particular structure

$$P \equiv S_1 \underset{L}{\bowtie} S_2$$

S_1, S_2 and L all restricted

- Quasi-reversibility

Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted direct interaction between components with a particular structure

$$P \equiv S_1 \underset{L}{\bowtie} S_2$$

S_1, S_2 and L all restricted

- Quasi-reversibility
- Reversibility

Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted direct interaction between components with a particular structure

$$P \equiv S_1 \underset{L}{\bowtie} S_2$$

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

$$P \equiv S_1 \underset{L}{\bowtie} S_2$$

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

$$P \equiv (S_1 \parallel S_2) \underset{L}{\bowtie} R$$

L and R restricted (wrt S_1 and S_2)

Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted direct interaction between components with a particular structure

$$P \equiv S_1 \underset{L}{\bowtie} S_2$$

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

$$P \equiv (S_1 \parallel S_2) \underset{L}{\bowtie} R$$

L and R restricted (wrt S_1 and S_2)

- Boucherie resource contention

Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted direct interaction between components with a particular structure

$$P \equiv S_1 \underset{L}{\bowtie} S_2$$

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

$$P \equiv (S_1 \parallel S_2) \underset{L}{\bowtie} R$$

L and R restricted (wrt S_1 and S_2)

- Boucherie resource contention
- Queueing discipline models

Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted direct interaction between components with a particular structure

$$P \equiv S_1 \underset{L}{\bowtie} S_2$$

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

$$P \equiv (S_1 \parallel S_2) \underset{L}{\bowtie} R$$

L and R restricted (wrt S_1 and S_2)

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

Outline

Other decomposition results

Aside from the work on product forms, Mertsiotakis and co-authors studied various forms of decomposed solution of SPA models 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.

Other decomposition results

Aside from the work on product forms, Mertsiotakis and co-authors studied various forms of decomposed solution of SPA models 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.

Approximate solutions

As we have seen, the numerical solution of the CTMC is seen as being the exact result.

Approximate solutions

As we have seen, the numerical solution of the CTMC is seen as being the exact result.

However due to the difficulties of staying within the confines of the Markov property most aggregation and decomposition techniques are not exact. They are nevertheless useful.

Approximate solutions

As we have seen, the numerical solution of the CTMC is seen as being the exact result.

However due to the difficulties of staying within the confines of the Markov property most aggregation and decomposition techniques are not exact. They are nevertheless useful.

Analysing a model via simulation can also produce useful results, often with some measure of confidence in the results with respect to the exact solution. This is achieved by repeatedly taking random walks through the state space and observing the results.

Approximate solutions

As we have seen, the numerical solution of the CTMC is seen as being the exact result.

However due to the difficulties of staying within the confines of the Markov property most aggregation and decomposition techniques are not exact. They are nevertheless useful.

Analysing a model via simulation can also produce useful results, often with some measure of confidence in the results with respect to the exact solution. This is achieved by repeatedly taking random walks through the state space and observing the results.

However, to conclude we are going to consider an approach which appears much more obviously approximate as it shifts our view of the system from discrete state to continuous.

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

- 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

Differential equations from PEPA models

- 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).

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

Differential equations from PEPA models

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

Consider the change in a small time δt:

$$N(\mathcal{C}_{i_j}, t+\delta t) - N(\mathcal{C}_{i_j}, t) = \underbrace{- \sum_{(\alpha,r)\in Ex(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{exit activities}} + \underbrace{\sum_{(\alpha,r)\in En(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{entry activities}}$$

Differential equations from PEPA models

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

Consider the change in a small time δt:

$$N(\mathcal{C}_{i_j}, t+\delta t) - N(\mathcal{C}_{i_j}, t) = \underbrace{- \sum_{(\alpha,r)\in Ex(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{exit activities}} + \underbrace{\sum_{(\alpha,r)\in En(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{entry activities}}$$

Differential equations from PEPA models

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

Consider the change in a small time δt:

$$N(\mathcal{C}_{i_j}, t+\delta t) - N(\mathcal{C}_{i_j}, t) = \underbrace{- \sum_{(\alpha,r)\in Ex(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{exit activities}} + \underbrace{\sum_{(\alpha,r)\in En(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{entry activities}}$$

Differential equations from PEPA models

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

Consider the change in a small time δt:

$$N(\mathcal{C}_{i_j}, t+\delta t) - N(\mathcal{C}_{i_j}, t) = \underbrace{- \sum_{(\alpha,r)\in Ex(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{exit activities}} + \underbrace{\sum_{(\alpha,r)\in En(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{entry activities}}$$

Differential equations from PEPA models

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

Consider the change in a small time δt:

$$N(\mathcal{C}_{i_j}, t+\delta t) - N(\mathcal{C}_{i_j}, t) = \underbrace{- \sum_{(\alpha,r)\in Ex(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{exit activities}} + \underbrace{\sum_{(\alpha,r)\in En(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{entry activities}}$$

Differential equations from PEPA models

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

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

$$\frac{dN(C_{i_j}, t)}{dt} = - \sum_{(\alpha, r) \in Ex(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l} \in pre(\alpha, r)} (N(C_{k_l}, t)) + \sum_{(\alpha, r) \in En(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l} \in pre(\alpha, r)} (N(C_{k_l}, t))$$

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.

Modelling with quantified process algebras

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Modelling with quantified process algebras

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

This example defines a system with nine reachable states:

1. $P_1 \parallel P_1$
2. $P_1 \parallel P_2$
3. $P_1 \parallel P_3$
4. $P_2 \parallel P_1$
5. $P_2 \parallel P_2$
6. $P_2 \parallel P_3$
7. $P_3 \parallel P_1$
8. $P_3 \parallel P_2$
9. $P_3 \parallel P_3$

The transitions between states have quantified duration r which can be evaluated against a CTMC or ODE interpretation.

Analysis based on Continuous-time Markov Chains

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 0$:

1. 1.0000
2. 0.0000
3. 0.0000
4. 0.0000
5. 0.0000
6. 0.0000
7. 0.0000
8. 0.0000
9. 0.0000

Analysis based on Continuous-time Markov Chains

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 1$:

1. 0.1642
2. 0.1567
3. 0.0842
4. 0.1567
5. 0.1496
6. 0.0804
7. 0.0842
8. 0.0804
9. 0.0432

Analysis based on Continuous-time Markov Chains

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 2$:

1. 0.1056
2. 0.1159
3. 0.1034
4. 0.1159
5. 0.1272
6. 0.1135
7. 0.1034
8. 0.1135
9. 0.1012

Analysis based on Continuous-time Markov Chains

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 3$:

1. 0.1082
2. 0.1106
3. 0.1100
4. 0.1106
5. 0.1132
6. 0.1125
7. 0.1100
8. 0.1125
9. 0.1119

Analysis based on Continuous-time Markov Chains

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 4$:

1. 0.1106
2. 0.1108
3. 0.1111
4. 0.1108
5. 0.1110
6. 0.1113
7. 0.1111
8. 0.1113
9. 0.1116

Analysis based on Continuous-time Markov Chains

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 5$:

1. 0.1111
2. 0.1110
3. 0.1111
4. 0.1110
5. 0.1110
6. 0.1111
7. 0.1111
8. 0.1111
9. 0.1111

Analysis based on Continuous-time Markov Chains

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 6$:

1. 0.1111
2. 0.1111
3. 0.1111
4. 0.1111
5. 0.1110
6. 0.1111
7. 0.1111
8. 0.1111
9. 0.1111

Analysis based on Continuous-time Markov Chains

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 7$:

1. 0.1111
2. 0.1111
3. 0.1111
4. 0.1111
5. 0.1111
6. 0.1111
7. 0.1111
8. 0.1111
9. 0.1111

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 0$:

P_1	2.0000
P_2	0.0000
P_3	0.0000

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 1$:

P_1	0.8121
P_2	0.7734
P_3	0.4144

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 2$:

P_1	0.6490
P_2	0.7051
P_3	0.6457

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 3$:

P_1	0.6587
P_2	0.6719
P_3	0.6692

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 4$:

P_1	0.6648
P_2	0.6665
P_3	0.6685

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 5$:

P_1	0.6666
P_2	0.6663
P_3	0.6669

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 6$:

P_1	0.6666
P_2	0.6666
P_3	0.6666

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 7$:

P_1	0.6666
P_2	0.6666
P_3	0.6666

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 0$:

P_1	3.0000
P_2	0.0000
P_3	0.0000

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 1$:	P_1	1.1782
	P_2	1.1628
	P_3	0.6590

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 2$:

P_1	0.9766
P_2	1.0754
P_3	0.9479

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 3$:

P_1	0.9838
P_2	1.0142
P_3	1.0020

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 4$:

P_1	0.9981
P_2	0.9995
P_3	1.0023

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 5$:

P_1	1.0001
P_2	0.9996
P_3	1.0003

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 6$:

P_1	1.0001
P_2	0.9999
P_3	1.0000

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 7$:

P_1	1.0000
P_2	0.9999
P_3	0.9999

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 8$:

P_1	1.0000
P_2	1.0000
P_3	1.0000

Isn't this just the Chapman-Kolmogorov equations?

It is possible to perform transient analysis of a continuous-time Markov chain by solving the Chapman-Kolmogorov differential equations:

$$\frac{d\pi(t)}{dt} = \pi(t)Q$$

[Stewart, 1994]

Isn't this just the Chapman-Kolmogorov equations?

It is possible to perform transient analysis of a continuous-time Markov chain by solving the Chapman-Kolmogorov differential equations:

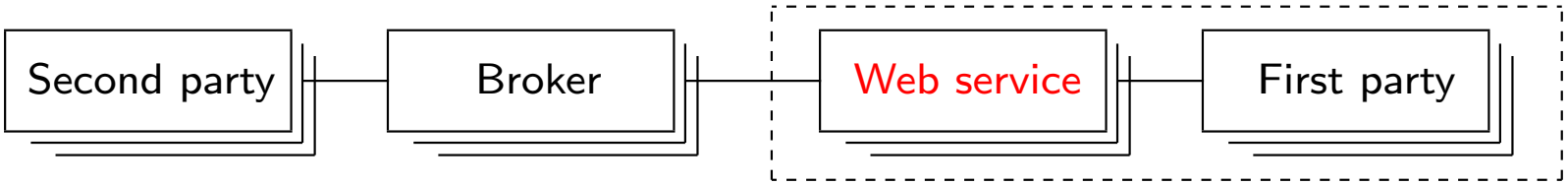
$$\frac{d\pi(t)}{dt} = \pi(t)Q$$

[Stewart, 1994]

That's not what we're doing. We go directly to ODEs.

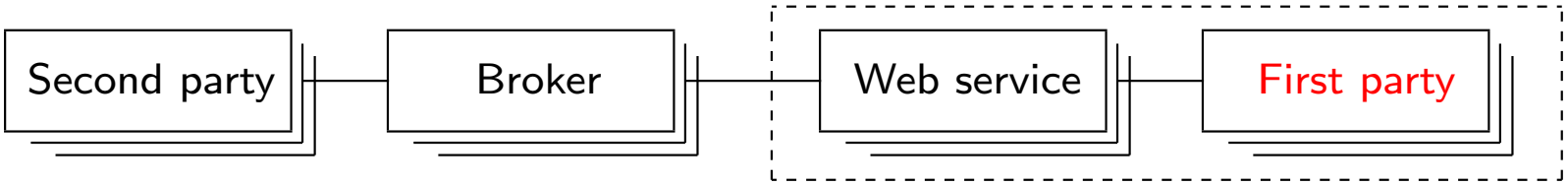
Outline

Example: Secure Web Service use



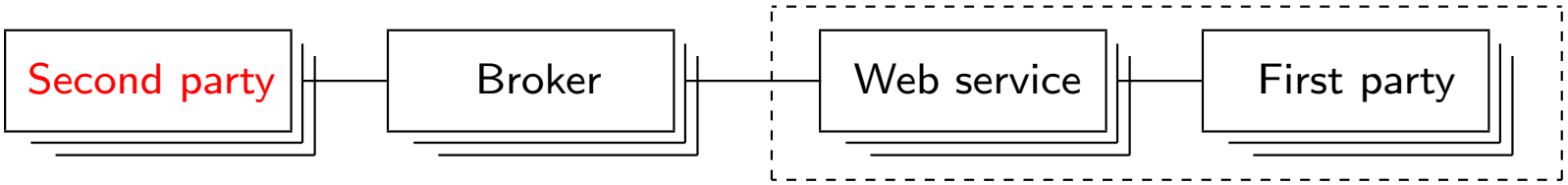
- The example which we consider is a Web service which has two types of clients:
  - first party application clients which access the web service across a secure intranet, and
  - second party browser clients which access the Web service across the Internet.
- Second party clients route their service requests via trusted brokers.

Example: Secure Web Service use



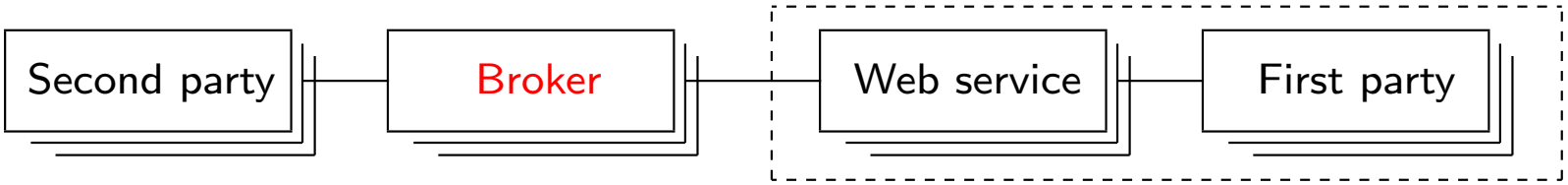
- The example which we consider is a Web service which has two types of clients:
  - first party application clients which access the web service across a secure intranet, and
  - second party browser clients which access the Web service across the Internet.
- Second party clients route their service requests via trusted brokers.

Example: Secure Web Service use



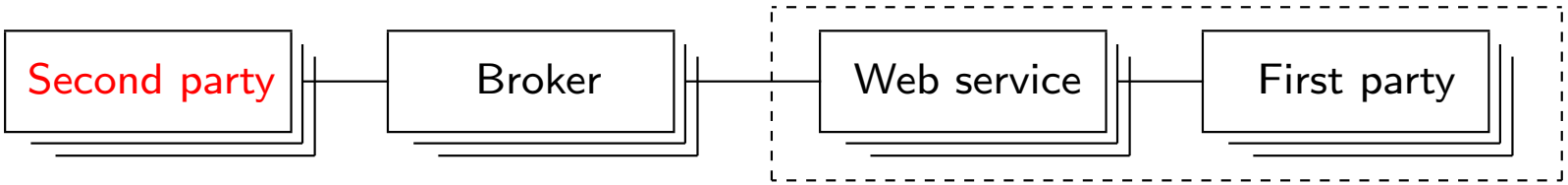
- The example which we consider is a Web service which has two types of clients:
  - first party application clients which access the web service across a secure intranet, and
  - second party browser clients which access the Web service across the Internet.
- Second party clients route their service requests via trusted brokers.

Example: Secure Web Service use



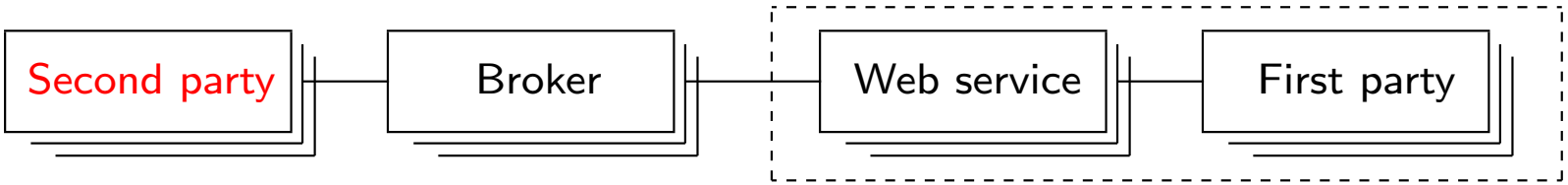
- The example which we consider is a Web service which has two types of clients:
  - first party application clients which access the web service across a secure intranet, and
  - second party browser clients which access the Web service across the Internet.
- Second party clients route their service requests via trusted brokers.

PEPA model: Second party clients and Brokers



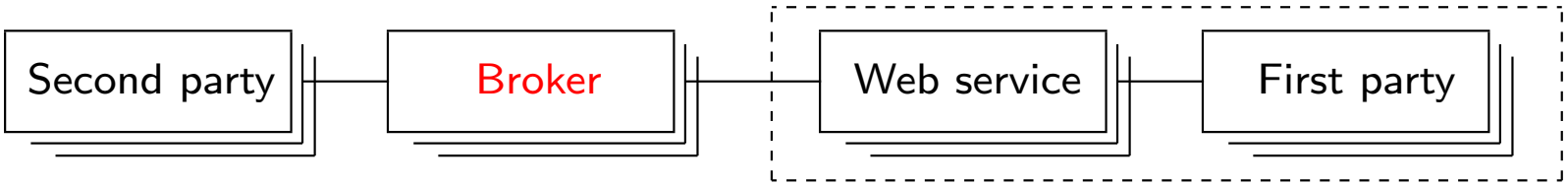
- A second party client composes service requests, encrypts these and sends them to its broker.

PEPA model: Second party clients and Brokers



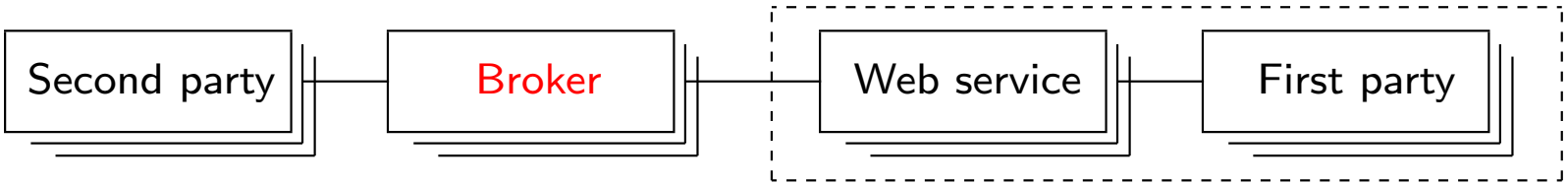
- A second party client composes service requests, encrypts these and sends them to its broker.
- It then waits for a response from the broker.

PEPA model: Second party clients and Brokers



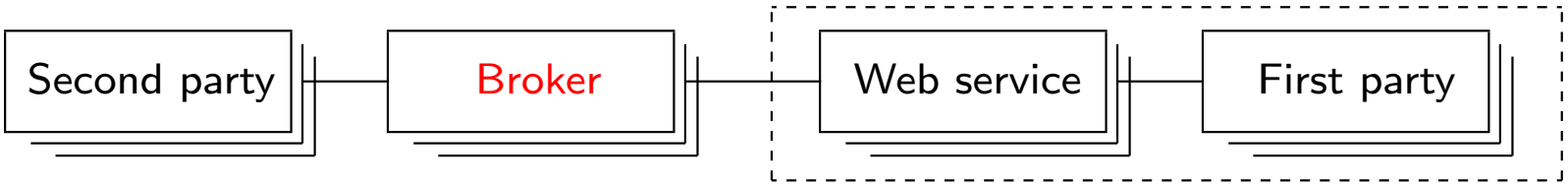
- A second party client composes service requests, encrypts these and sends them to its broker.
- It then waits for a response from the broker.
- Brokers add decryption and encryption steps to build end-to-end security from point-to-point security.

PEPA model: Second party clients and Brokers



- A second party client composes service requests, encrypts these and sends them to its broker.
- It then waits for a response from the broker.
- Brokers add decryption and encryption steps to build end-to-end security from point-to-point security.
- The broker forwards the request to the Web service and then waits for a response.

PEPA model: Second party clients and Brokers



- A second party client composes service requests, encrypts these and sends them to its broker.
- It then waits for a response from the broker.
- Brokers add decryption and encryption steps to build end-to-end security from point-to-point security.
- The broker forwards the request to the Web service and then waits for a response.
- Decryption and re-encrytion are performed before returning the response to the client.

Cost of analysis

- We compare ODE-based evaluation against other techniques which could be used to analyse the model.

Cost of analysis

- We compare ODE-based evaluation against other techniques which could be used to analyse the model.
  - Steady-state and transient analysis as implemented by the PRISM probabilistic model-checker.

Cost of analysis

- We compare ODE-based evaluation against other techniques which could be used to analyse the model.
  - Steady-state and transient analysis as implemented by the PRISM probabilistic model-checker.
  - Monte Carlo Markov Chain simulation (a Java implementation of Gillespie's Direct Method).

Running times from analyses (in seconds)

Second party clients	Brokers	Web service instances	First party clients	Number of states in the full state-space	Number of states in the aggregated state-space	Sparse matrix steady-state	Matrix/MTBDD steady-state	Transient solution for time $t = 100$	MCMC simulation one run to $t = 100$	ODE solution
1	1	1	1	48	48	1.04	1.10	1.01	2.47	2.81

Running times from analyses (in seconds)

Second party clients	Brokers	Web service instances	First party clients	Number of states in the full state-space	Number of states in the aggregated state-space	Sparse matrix steady-state	Matrix/MTBDD steady-state	Transient solution for time $t = 100$	MCMC simulation one run to $t = 100$	ODE solution
1	1	1	1	48	48	1.04	1.10	1.01	2.47	2.81
2	2	2	2	6,304	860	2.15	2.26	2.31	2.45	2.81

Running times from analyses (in seconds)

Second party clients	Brokers	Web service instances	First party clients	Number of states in the full state-space	Number of states in the aggregated state-space	Sparse matrix steady-state	Matrix/MTBDD steady-state	Transient solution for time $t = 100$	MCMC simulation one run to $t = 100$	ODE solution
1	1	1	1	48	48	1.04	1.10	1.01	2.47	2.81
2	2	2	2	6,304	860	2.15	2.26	2.31	2.45	2.81
3	3	3	3	1,130,496	161,296	172.48	255.48	588.80	2.48	2.83

Running times from analyses (in seconds)

Second party clients	Brokers	Web service instances	First party clients	Number of states in the full state-space	Number of states in the aggregated state-space	Sparse matrix steady-state	Matrix/MTBDD steady-state	Transient solution for time $t = 100$	MCMC simulation one run to $t = 100$	ODE solution
1	1	1	1	48	48	1.04	1.10	1.01	2.47	2.81
2	2	2	2	6,304	860	2.15	2.26	2.31	2.45	2.81
3	3	3	3	1,130,496	161,296	172.48	255.48	588.80	2.48	2.83
4	4	4	4	>234M	–	–	–	–	2.44	2.85

Running times from analyses (in seconds)

Second party clients	Brokers	Web service instances	First party clients	Number of states in the full state-space	Number of states in the aggregated state-space	Sparse matrix steady-state	Matrix/MTBDD steady-state	Transient solution for time $t = 100$	MCMC simulation one run to $t = 100$	ODE solution
1	1	1	1	48	48	1.04	1.10	1.01	2.47	2.81
2	2	2	2	6,304	860	2.15	2.26	2.31	2.45	2.81
3	3	3	3	1,130,496	161,296	172.48	255.48	588.80	2.48	2.83
4	4	4	4	>234M	–	–	–	–	2.44	2.85
100	100	100	100	–	–	–	–	–	2.78	2.78

Running times from analyses (in seconds)

Second party clients	Brokers	Web service instances	First party clients	Number of states in the full state-space	Number of states in the aggregated state-space	Sparse matrix steady-state	Matrix/MTBDD steady-state	Transient solution for time $t = 100$	MCMC simulation one run to $t = 100$	ODE solution
1	1	1	1	48	48	1.04	1.10	1.01	2.47	2.81
2	2	2	2	6,304	860	2.15	2.26	2.31	2.45	2.81
3	3	3	3	1,130,496	161,296	172.48	255.48	588.80	2.48	2.83
4	4	4	4	>234M	–	–	–	–	2.44	2.85
100	100	100	100	–	–	–	–	–	2.78	2.78
1000	100	500	1000	–	–	–	–	–	3.72	2.77

Running times from analyses (in seconds)

Second party clients	Brokers	Web service instances	First party clients	Number of states in the full state-space	Number of states in the aggregated state-space	Sparse matrix steady-state	Matrix/MTBDD steady-state	Transient solution for time $t = 100$	MCMC simulation one run to $t = 100$	ODE solution
1	1	1	1	48	48	1.04	1.10	1.01	2.47	2.81
2	2	2	2	6,304	860	2.15	2.26	2.31	2.45	2.81
3	3	3	3	1,130,496	161,296	172.48	255.48	588.80	2.48	2.83
4	4	4	4	>234M	–	–	–	–	2.44	2.85
100	100	100	100	–	–	–	–	–	2.78	2.78
1000	100	500	1000	–	–	–	–	–	3.72	2.77
1000	1000	1000	1000	–	–	–	–	–	5.44	2.77

Running times from analyses (in seconds)

Second party clients	Brokers	Web service instances	First party clients	Number of states in the full state-space	Number of states in the aggregated state-space	Sparse matrix steady-state	Matrix/MTBDD steady-state	Transient solution for time $t = 100$	MCMC simulation one run to $t = 100$	ODE solution
1	1	1	1	48	48	1.04	1.10	1.01	2.47	2.81
2	2	2	2	6,304	860	2.15	2.26	2.31	2.45	2.81
3	3	3	3	1,130,496	161,296	172.48	255.48	588.80	2.48	2.83
4	4	4	4	>234M	–	–	–	–	2.44	2.85
100	100	100	100	–	–	–	–	–	2.78	2.78
1000	100	500	1000	–	–	–	–	–	3.72	2.77
1000	1000	1000	1000	–	–	–	–	–	5.44	2.77

Time series analysis via ODEs

- We assume a system in which the number of clients of both kinds, brokers, and web service instances are all 1000.

Time series analysis via ODEs

- We assume a system in which the number of clients of both kinds, brokers, and web service instances are all 1000.
- We present the results from our ODE integrator as time-series plots of the number of each type of component behaviour as a function of time, as time runs from $t = 0$ to $t = 100$.

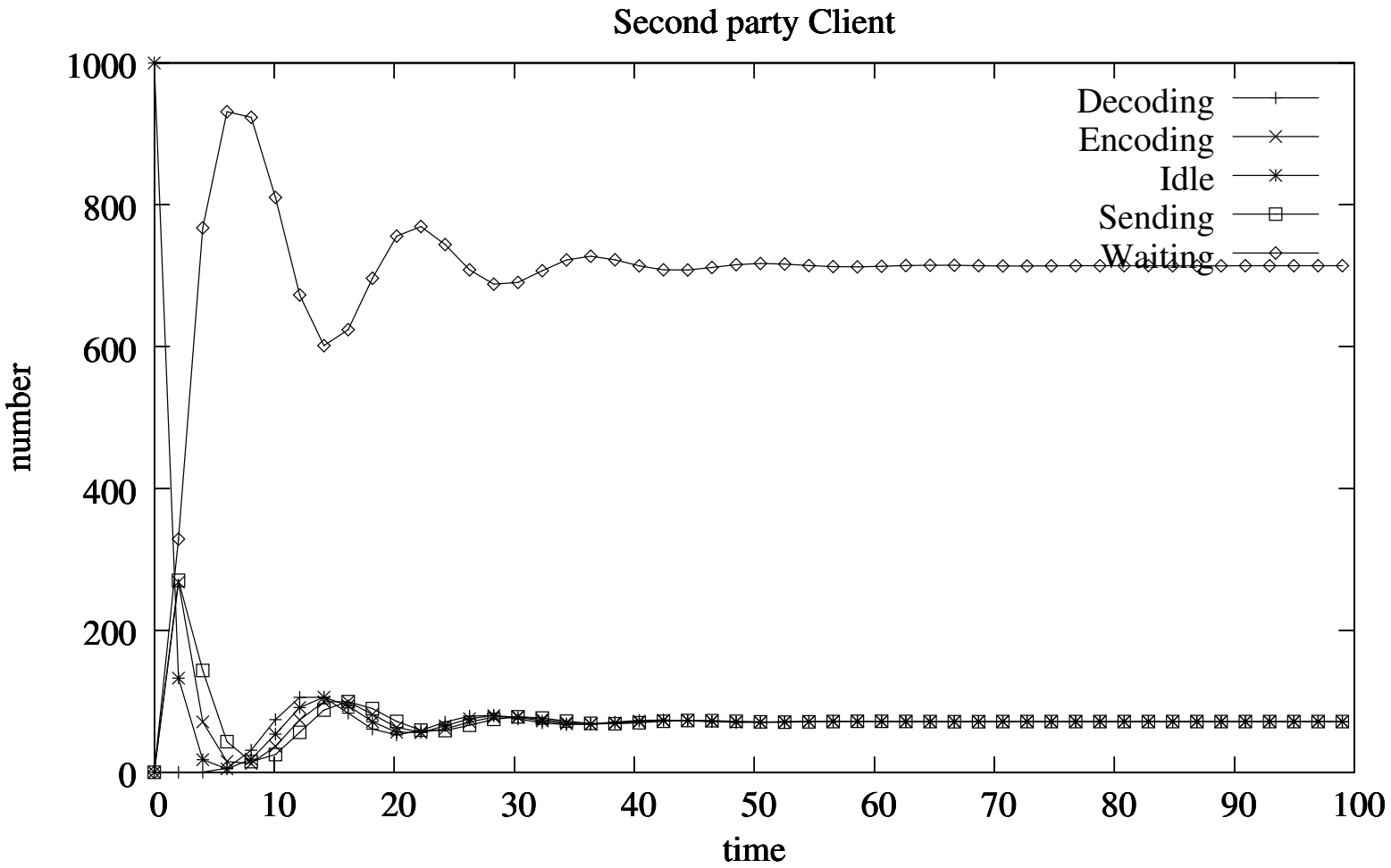
Time series analysis via ODEs

- We assume a system in which the number of clients of both kinds, brokers, and web service instances are all 1000.
- We present the results from our ODE integrator as time-series plots of the number of each type of component behaviour as a function of time, as time runs from $t = 0$ to $t = 100$.
- The graphs show fluctuations in the numbers of components with respect to time.

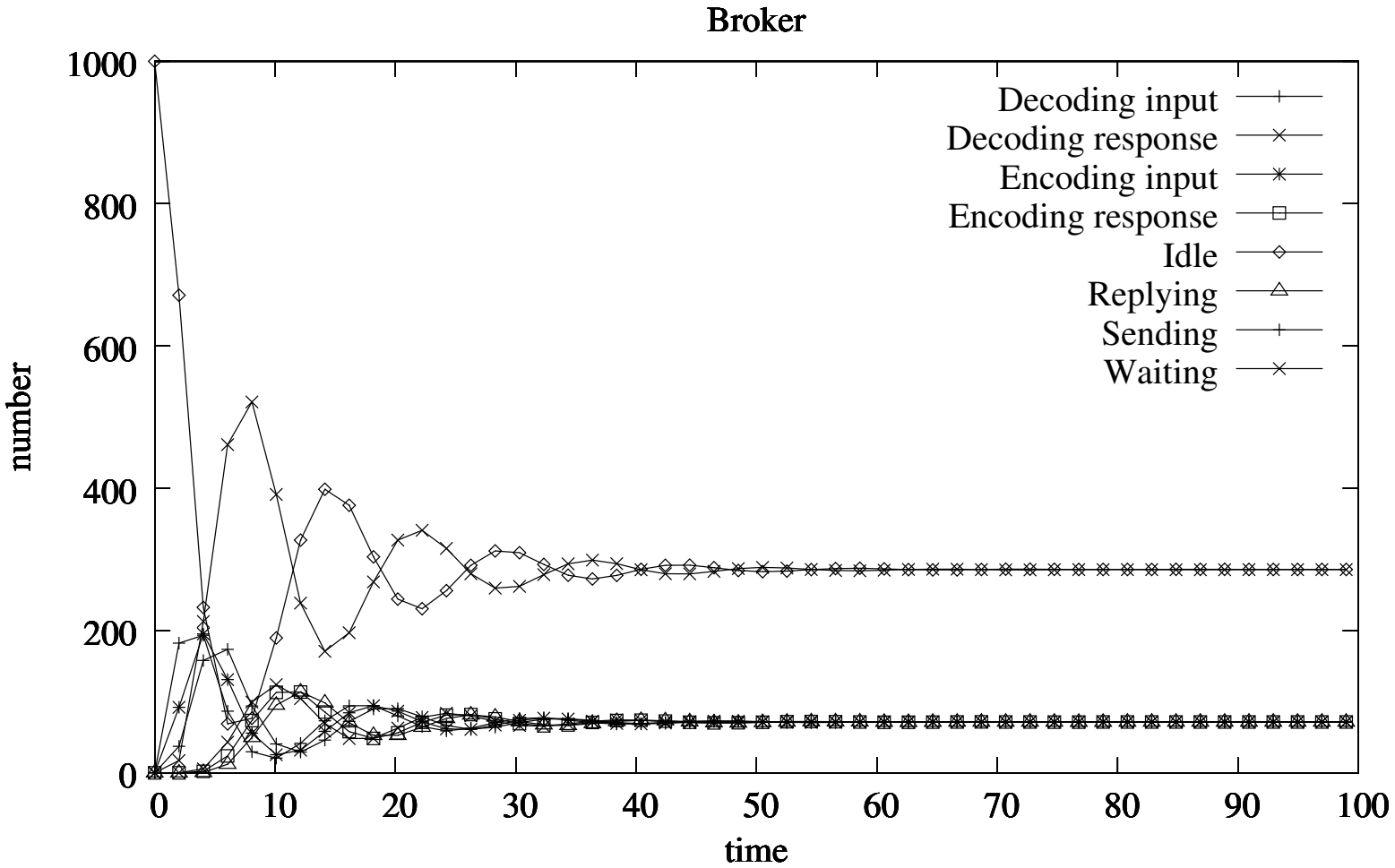
Time series analysis via ODEs

- We assume a system in which the number of clients of both kinds, brokers, and web service instances are all 1000.
- We present the results from our ODE integrator as time-series plots of the number of each type of component behaviour as a function of time, as time runs from $t = 0$ to $t = 100$.
- The graphs show fluctuations in the numbers of components with respect to time.
- We can observe an initial flurry of activity until the system stabilises into its steady-state equilibrium at time (around) $t = 50$.

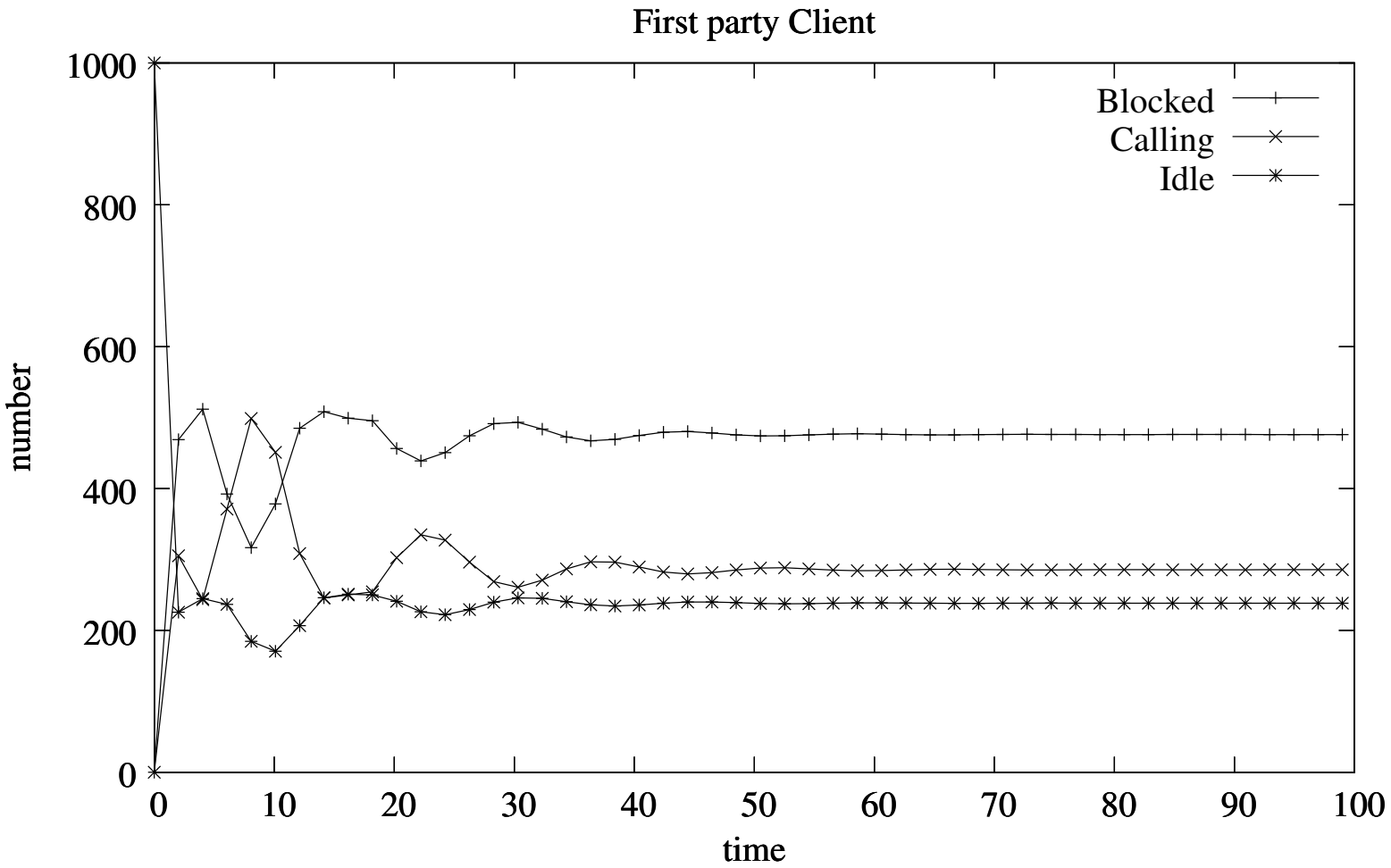
Second party clients



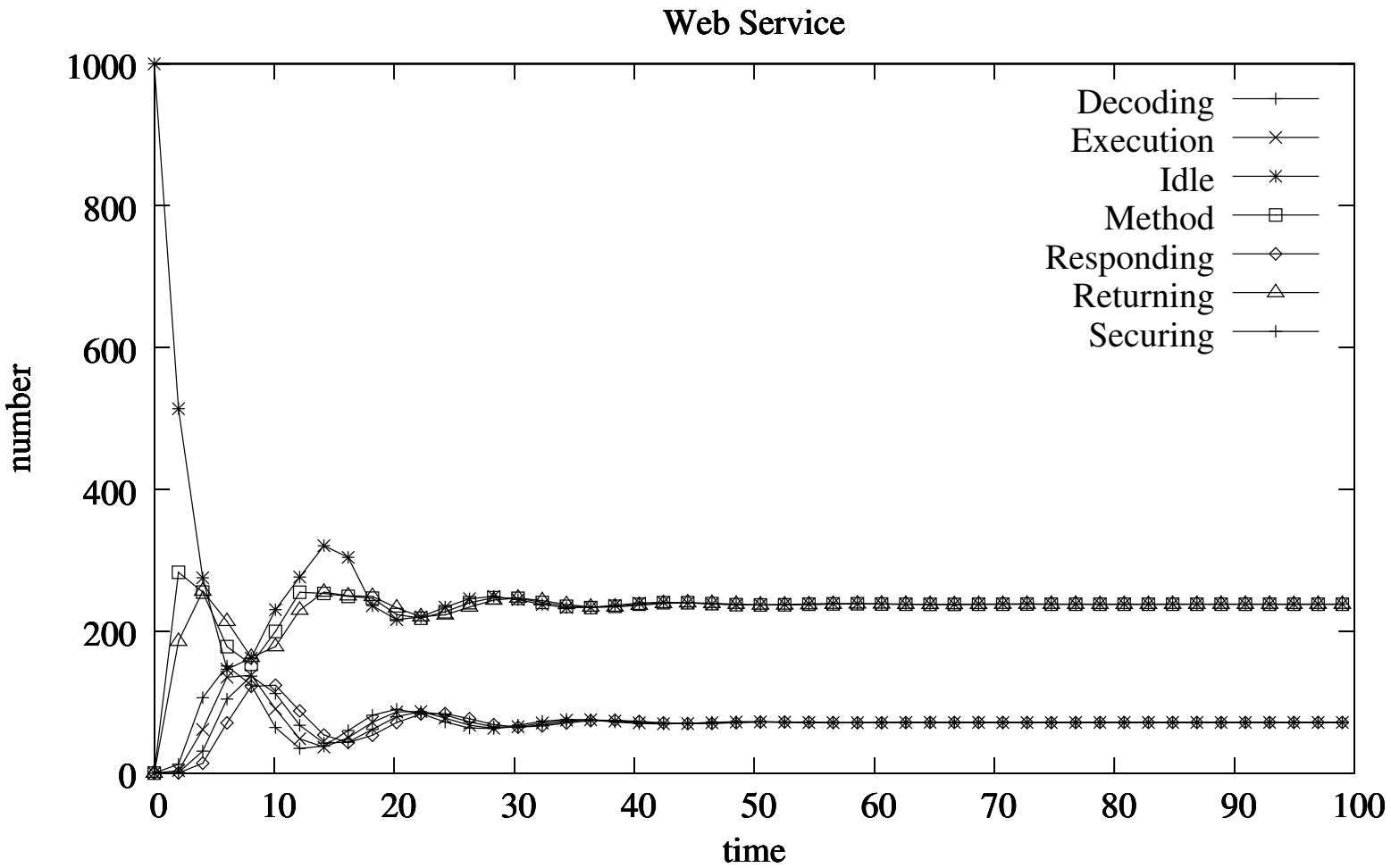
Brokers


Broker
number
time
0
200
400
600
800
1000
0
10
20
30
40
50
60
70
80
90
100
Decoding input
Decoding response
Encoding input
Encoding response
Idle
Replying
Sending
Waiting

First party clients



Web service


Web Service
number
time
0
200
400
600
800
1000
0
10
20
30
40
50
60
70
80
90
100
Decoding
Execution
Idle
Method
Responding
Returning
Securing

Comparison with CTMC empirically

- Direct comparison of results is not possible.

Comparison with CTMC empirically

- Direct comparison of results is not possible.
- For systems with the same blocking characteristics (1000 instances of all components vs 1 instance of each component) we compared values of performance measures derived via each of the two models.

Comparison with CTMC empirically

- Direct comparison of results is not possible.
- For systems with the same blocking characteristics (1000 instances of all components vs 1 instance of each component) we compared values of performance measures derived via each of the two models.
- We found good agreement between the CTMC results and those results obtained by ODE solution after the system stabilises into its steady-state equilibrium.

Comparison with CTMC empirically

- Direct comparison of results is not possible.
- For systems with the same blocking characteristics (1000 instances of all components vs 1 instance of each component) we compared values of performance measures derived via each of the two models.
- We found good agreement between the CTMC results and those results obtained by ODE solution after the system stabilises into its steady-state equilibrium. Less than 1% difference.

Comparison with CTMC theoretically

Recent results by Jie Ding [PhD thesis, Edinburgh 2010] have established the conditions under which the equilibrium reached by the ODEs derived from a PEPA model coincide with the steady state probability distribuion of the CTMC derived by the SOS semantics.

Comparison with CTMC theoretically

Recent results by Jie Ding [PhD thesis, Edinburgh 2010] have established the conditions under which the equilibrium reached by the ODEs derived from a PEPA model coincide with the steady state probability distribuion of the CTMC derived by the SOS semantics.

Unfortunately these results involved testing conditions of the eigenvalues of a matrix associated with the ODEs and is not readily checked at the syntactically level.

Comparison with CTMC theoretically

Recent results by Jie Ding [PhD thesis, Edinburgh 2010] have established the conditions under which the equilibrium reached by the ODEs derived from a PEPA model coincide with the steady state probability distribuion of the CTMC derived by the SOS semantics.

Unfortunately these results involved testing conditions of the eigenvalues of a matrix associated with the ODEs and is not readily checked at the syntactically level.

We are working on finding a syntactic characterisation.

Some 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*, FMPA 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.

Thank you