Fluid Approximation for the Analysis of Collective Systems

Jane Hillston
LFCS, University of Edinburgh

July 23rd 2014
1 Introduction
   - Collective Systems
   - Quantitative Analysis
   - Stochastic Process Algebra

2 Quantitative Analysis of Collective Systems
   - Model construction
   - Mathematical analysis: fluid approximation
   - Numerical illustration
   - Deriving properties: fluid model checking

3 Conclusions
1 Introduction
   ■ Collective Systems
   ■ Quantitative Analysis
   ■ Stochastic Process Algebra
2 Quantitative Analysis of Collective Systems
   ■ Model construction
   ■ Mathematical analysis: fluid approximation
   ■ Numerical illustration
   ■ Deriving properties: fluid model checking
3 Conclusions
We are surrounded by examples of collective systems:
We are surrounded by examples of collective systems:
in the natural world ....
Collective Systems

We are surrounded by examples of collective systems:

.... and in the man-made world
We are surrounded by examples of collective systems:

.... and in the man-made world
We are surrounded by examples of collective systems:
an informatic environment
Collective Systems

From a computer science perspective these systems can be viewed as being made up of a large number of interacting entities.

Each entity may have its own properties, objectives and actions.

At the system level these combine to create the collective behaviour.
Collective Systems

The behaviour of the system is thus dependent on the behaviour of the individual entities.
Collective Systems

The behaviour of the system is thus dependent on the behaviour of the individual entities.
Collective Systems

The behaviour of the system is thus dependent on the behaviour of the individual entities.

And the behaviour of the individuals will be influenced by the state of the overall system.
Robin Milner coined the term of **informatics environment**, in which pervasive computing elements are embedded in the human environment, invisibly providing services and responding to requirements.
Robin Milner coined the term of *informatics environment*, in which pervasive computing elements are embedded in the human environment, invisibly providing services and responding to requirements.

Such systems are now becoming the reality, and many form collective adaptive systems, in which large numbers of computing elements collaborate to meet the human need.
Robin Milner coined the term of informatics environment, in which pervasive computing elements are embedded in the human environment, invisibly providing services and responding to requirements.

Such systems are now becoming the reality, and many form collective adaptive systems, in which large numbers of computing elements collaborate to meet the human need.

For instance, may examples of such systems can be found in components of Smart Cities, such as smart urban transport and smart grid electricity generation and storage.
Quantitative Modelling

Performance modelling aims to construct models of the dynamic behaviour of systems in order to support the efficient and equitable sharing of resources.
Performance modelling aims to construct models of the dynamic behaviour of systems in order to support the efficient and equitable sharing of resources.

Markovian-based discrete event models have been applied to computer systems since the mid-1960s and communication systems since the early 20th century.
Quantitative Modelling

Performance modelling aims to construct models of the dynamic behaviour of systems in order to support the efficient and equitable sharing of resources.

Markovian-based discrete event models have been applied to computer systems since the mid-1960s and communication systems since the early 20th century.

Originally queueing networks were primarily used to construct models, and sophisticated analysis techniques were developed.
Performance modelling aims to construct models of the dynamic behaviour of systems in order to support the efficient and equitable sharing of resources.

Markovian-based discrete event models have been applied to computer systems since the mid-1960s and communication systems since the early 20th century.

Originally queueing networks were primarily used to construct models, and sophisticated analysis techniques were developed.

These techniques are no longer widely applicable for expressing the dynamic behaviour observed in distributed systems, and this is even more true of systems with collective behaviour.
Performance Modelling: Motivation

Capacity Planning

- How many clients can the existing server support and maintain reasonable response times?
- How many buses do I need to maintain service at peak time in a smart urban transport system?
Performance Modelling: Motivation

Capacity Planning
- How many clients can the existing server support and maintain reasonable response times?
- How many buses do I need to maintain service at peak time in a smart urban transport system?

System Configuration
- Where should I place base stations in a network to keep blocking probabilities low?
- What capacity do I need at bike stations to minimise the movement of bikes by truck?
Performance Modelling: Motivation

Capacity Planning
- How many clients can the existing server support and maintain reasonable response times?
- How many buses do I need to maintain service at peak time in a smart urban transport system?

System Configuration
- Where should I place base stations in a network to keep blocking probabilities low?
- What capacity do I need at bike stations to minimise the movement of bikes by truck?

System Tuning
- In an automated factory what speed of conveyor belt will minimize robot idle time and jamming but maximize throughput?
- What strategy can I use to maintain supply-demand balance within a smart electricity grid?
Performance Modelling

The size and complexity of real systems makes the direct construction of discrete state models costly and error-prone.
Performance Modelling

The size and complexity of real systems makes the direct construction of discrete state models costly and error-prone.

For the last three decades there has been substantial interest in applying formal modelling techniques enhanced with information about timing and probability.
Performance Modelling

The size and complexity of real systems makes the direct construction of discrete state models costly and error-prone.

For the last three decades there has been substantial interest in applying formal modelling techniques enhanced with information about timing and probability.

From these high-level system descriptions the underlying mathematical model (Continuous Time Markov Chain (CTMC)) can be automatically generated.
Performance Modelling

The size and complexity of real systems makes the direct construction of discrete state models costly and error-prone.

For the last three decades there has been substantial interest in applying formal modelling techniques enhanced with information about timing and probability.

From these high-level system descriptions the underlying mathematical model (Continuous Time Markov Chain (CTMC)) can be automatically generated.

Primary examples include:
- Stochastic Petri Nets and
- Stochastic/Markovian Process Algebras.
Process Algebra

- Models consist of agents which engage in actions.

\[ \alpha \cdot P \]

- action type or name
- agent/component

\[ a \quad c \quad b \]

Labelled transition system - SOS rules
Models consist of agents which engage in actions.
Models consist of agents which engage in actions.

\[ \alpha.P \]

- \( \alpha \): action type or name
- \( P \): agent/component

Diagram:

- Circle with arrows connecting 'a', 'b', and 'c'
- Arrows indicate interactions among the agents.
- Models consist of **agents** which engage in **actions**.

\[ \alpha.P \]

- Action type or name
- Agent/component

![Diagram](image-url)


- Models consist of agents which engage in actions.

\[ \alpha . P \]

- action type or name
- agent/component

![Diagram of a process algebra model with agents a, b, and c interacting]

- Labelled transition system - SOS rules
■ Models consist of agents which engage in actions.

\[ \alpha.P \]

- action type or name
- agent/component

■ The structured operational (interleaving) semantics of the language is used to generate a labelled transition system.
Process Algebra

- Models consist of agents which engage in actions.

\[ \alpha \cdot P \]

- The structured operational (interleaving) semantics of the language is used to generate a labelled transition system.

Process algebra model
Process Algebra

- Models consist of agents which engage in actions.

\[ \alpha.P \]

- The structured operational (interleaving) semantics of the language is used to generate a labelled transition system.

Process algebra model \[\rightarrow\] SOS rules
Process Algebra

- Models consist of agents which engage in actions.

\[ \alpha.P \]

- The structured operational (interleaving) semantics of the language is used to generate a labelled transition system.

Process algebra model  \(\xrightarrow{\text{SOS rules}}\) Labelled transition system
Process Algebra

- Models consist of agents which engage in actions.

\[ \alpha . P \]

- The structured operational (interleaving) semantics of the language is used to generate a labelled transition system.

Process algebra model \(\rightarrow\) SOS rules \(\rightarrow\) Labelled transition system
Stochastic process algebras

Process algebras where models are decorated with quantitative information used to generate a stochastic process are **stochastic process algebras (SPA)**.
Models are constructed from components which engage in activities.

\[(\alpha, r).P\]

- action type or name
- activity rate (parameter of an exponential distribution)
- component/derivative
Stochastic Process Algebra

- Models are constructed from components which engage in activities.

\[(\alpha, r).P\]

- The language is used to generate a CTMC for performance modelling.
■ Models are constructed from components which engage in activities.

\[ (\alpha, r).P \]

- action type or name
- activity rate (parameter of an exponential distribution)
- component/derivative

■ The language is used to generate a CTMC for performance modelling.

SPA
MODEL
Models are constructed from components which engage in activities.

\[(\alpha, r).P\]

- action type or name
- component/derivative
- activity rate (parameter of an exponential distribution)

The language is used to generate a CTMC for performance modelling.

SPA MODEL SOS rules
Models are constructed from components which engage in activities.

The language is used to generate a CTMC for performance modelling.
Stochastic Process Algebra

- Models are constructed from components which engage in activities.

\[(\alpha, r).P\]

- The language is used to generate a CTMC for performance modelling.

SPA MODEL \(\xrightarrow{\text{SOS rules}}\) LABELLED TRANSITION SYSTEM \(\xrightarrow{\text{state transition diagram}}\)
Models are constructed from components which engage in activities.

\[(\alpha, r).P\]

The language is used to generate a CTMC for performance modelling.
Integrated analysis

Qualitative verification can now be complemented by quantitative verification.
Integrated analysis

**Qualitative** verification can now be complemented by **quantitative** verification.

Reachability analysis

How long will it take for the system to arrive in a particular state?
Integrated analysis

**Qualitative** verification can now be complemented by **quantitative** verification.

Does a given property $\phi$ hold within the system with a given probability?
Integrated analysis

Qualitative verification can now be complemented by quantitative verification.

Model checking

For a given starting state how long is it until a given property $\phi$ holds?
Performance Evaluation Process Algebra

\[(\alpha, f).P\]  Prefix

\[P_1 + P_2\]  Choice

\[P_1 \Box L P_2\]  Co-operation

\[P/L\]  Hiding

\[X\]  Constant
Performance Evaluation Process Algebra

\[(\alpha, f).P\] Prefix

\[P_1 + P_2\] Choice

\[P_1 \bowtie_l P_2\] Co-operation

\[P/L\] Hiding

\[X\] Constant
Performance Evaluation Process Algebra

\[(\alpha, f).P\] Prefix

\[P_1 + P_2\] Choice

\[P_1 \boxplus L P_2\] Co-operation

\[P/L\] Hiding

\[X\] Constant
Performance Evaluation Process Algebra

\[(\alpha, f).P\] Prefix

\[P_1 + P_2\] Choice

\[P_1 \bowtie P_2\] Co-operation

\[P/L\] Hiding

\[X\] Constant
Performance Evaluation Process Algebra

\[(\alpha, \mathbf{f}).P]\quad \text{Prefix}

\[P_1 + P_2\] \quad \text{Choice}

\[P_1 \parallel_l P_2\] \quad \text{Co-operation}

\[P/L\] \quad \text{Hiding}

\[X\] \quad \text{Constant}
Performance Evaluation Process Algebra

\((\alpha, f).P\)   \hspace{1cm} \text{Prefix}

\(P_1 + P_2\)   \hspace{1cm} \text{Choice}

\(\parallel_l P_1 \boxtimes P_2\)   \hspace{1cm} \text{Co-operation}

\(P/L\)   \hspace{1cm} \text{Hiding}

\(\times\)   \hspace{1cm} \text{Constant}
Performance Evaluation Process Algebra

\[ (\alpha, f).P \] Prefix
\[ P_1 + P_2 \] Choice
\[ P_1 \preceq P_2 \] Co-operation
\[ P/L \] Hiding
\[ X \] Constant

\[ P_1 \parallel P_2 \] is a derived form for \[ P_1 \bowtie P_2. \]
Performance Evaluation Process Algebra

\[(\alpha, f).P\] Prefix

\[P_1 + P_2\] Choice

\[P_1 \Leftrightarrow L P_2\] Co-operation

\[P/L\] Hiding

\[X\] Constant

\[P_1 \parallel P_2\] is a derived form for \[P_1 \triangleright \emptyset P_2\].

When working with large numbers of entities, we write \(P[n]\) to denote an array of \(n\) copies of \(P\) executing in parallel.

\[P[5] \equiv (P \parallel P \parallel P \parallel P \parallel P)\]
Structured Operational Semantics

PEPA is defined using a Plotkin-style structured operational semantics (a “small step” semantics).
PEPA is defined using a Plotkin-style structured operational semantics (a “small step” semantics).

Prefix

\[(\alpha, r).E \xrightarrow{(\alpha,r)} E\]
Structured Operational Semantics

PEPA is defined using a Plotkin-style structured operational semantics (a “small step” semantics).

**Prefix**

\[(\alpha, r).E \xrightarrow{(\alpha, r)} E\]

**Choice**

\[E \xrightarrow{(\alpha, r)} E'\]
\[E + F \xrightarrow{(\alpha, r)} E'\]

\[F \xrightarrow{(\alpha, r)} F'\]
\[E + F \xrightarrow{(\alpha, r)} F'\]
Structured Operational Semantics: Cooperation \((\alpha \notin L)\)

Cooperation

\[
\begin{align*}
    & E \xrightarrow{(\alpha, r)} E' \\
\iff & E \boxtimes L F \xrightarrow{(\alpha, r)} E' \boxtimes L F \\
\text{(}\alpha \notin L\text{)}
\end{align*}
\]
Structured Operational Semantics: Cooperation ($\alpha \notin L$)

Cooperation

\[
\begin{align*}
E \xrightarrow{\alpha,r} E' \\
E \bowtie_L F \xrightarrow{\alpha,r} E' \bowtie_L F
\end{align*}
\]

\[
\begin{align*}
F \xrightarrow{\alpha,r} F' \\
E \bowtie_L F \xrightarrow{\alpha,r} E \bowtie_L F'
\end{align*}
\]
Structured Operational Semantics: Cooperation \((\alpha \in L)\)

\[
\begin{array}{c}
\text{Cooperation} \\
E \xrightarrow{(\alpha, r_1)} E' \quad F \xrightarrow{(\alpha, r_2)} F' \\
E \underbrace{\boxtimes}_L F \xrightarrow{(\alpha, R)} E' \underbrace{\boxtimes}_L F'
\end{array}
\]
Structured Operational Semantics: Cooperation ($\alpha \in L$)

Cooperation

\[
\begin{align*}
E \xrightarrow{(\alpha, r_1)} E' \quad & \quad F \xrightarrow{(\alpha, r_2)} F' \\
E \not\leftrightarrow_L F \xrightarrow{(\alpha, R)} E' \not\leftrightarrow_L F'
\end{align*}
\]

where

\[
R = \frac{r_1}{r_\alpha(E)} \frac{r_2}{r_\alpha(F)} \min(r_\alpha(E), r_\alpha(F))
\]
Apparent Rate

\[ r_\alpha((\beta, r).P) = \begin{cases} r & \beta = \alpha \\ 0 & \beta \neq \alpha \end{cases} \]

\[ r_\alpha(P + Q) = r_\alpha(P) + r_\alpha(Q) \]

\[ r_\alpha(A) = r_\alpha(P) \quad \text{where } A \overset{\text{def}}{=} P \]

\[ r_\alpha(P \, \bowtie_{L} \, Q) = \begin{cases} r_\alpha(P) + r_\alpha(Q) & \alpha \notin L \\ \min(r_\alpha(P), r_\alpha(Q)) & \alpha \in L \end{cases} \]

\[ r_\alpha(P/L) = \begin{cases} r_\alpha(P) & \alpha \notin L \\ 0 & \alpha \in L \end{cases} \]
Structured Operational Semantics: Hiding

Hiding

\[
\begin{align*}
E \xrightarrow{(\alpha,r)} & E' \\
E/L \xrightarrow{(\alpha,r)} & E'/L \\
(\alpha \notin L)
\end{align*}
\]
Structured Operational Semantics: Hiding

Hiding

\[
\begin{align*}
E & \xrightarrow{(\alpha,r)} E' \quad (\alpha \notin L) \\
E/L & \xrightarrow{(\alpha,r)} E'/L
\end{align*}
\]

\[
\begin{align*}
E & \xrightarrow{(\alpha,r)} E' \quad (\alpha \in L) \\
E/L & \xrightarrow{(\tau,r)} E'/L
\end{align*}
\]
Structured Operational Semantics: Constants

\[
E \xrightarrow{(\alpha, r)} E' \quad (A \overset{\text{def}}{=} E)
\]

Constant
A simple example: processors and resources

\[
\begin{align*}
Proc_0 & \overset{\text{def}}{=} (\text{task}1, r_1).Proc_1 \\
Proc_1 & \overset{\text{def}}{=} (\text{task}2, r_2).Proc_0 \\
Res_0 & \overset{\text{def}}{=} (\text{task}1, r_3).Res_1 \\
Res_1 & \overset{\text{def}}{=} (\text{reset}, r_4).Res_0 \\
\end{align*}
\]

\[
\begin{align*}
Proc_0 & \parallel Res_0 \\
\end{align*}
\]
A simple example: processors and resources

\[
\begin{align*}
    \text{Proc}_0 & \overset{\text{def}}{=} (\text{task1}, r_1).\text{Proc}_1 \\
    \text{Proc}_1 & \overset{\text{def}}{=} (\text{task2}, r_2).\text{Proc}_0 \\
    \text{Res}_0 & \overset{\text{def}}{=} (\text{task1}, r_3).\text{Res}_1 \\
    \text{Res}_1 & \overset{\text{def}}{=} (\text{reset}, r_4).\text{Res}_0
\end{align*}
\]

\[
R = \min(r_1, r_3)
\]
A simple example: processors and resources

\[
\begin{align*}
\text{Proc}_0 & \overset{\text{def}}{=} (\text{task1}, r_1).\text{Proc}_1 \\
\text{Proc}_1 & \overset{\text{def}}{=} (\text{task2}, r_2).\text{Proc}_0 \\
\text{Res}_0 & \overset{\text{def}}{=} (\text{task1}, r_3).\text{Res}_1 \\
\text{Res}_1 & \overset{\text{def}}{=} (\text{reset}, r_4).\text{Res}_0
\end{align*}
\]

\[
\begin{array}{cccc}
-R & R & 0 & 0 \\
0 & -(r_2 + r_4) & r_4 & r_2 \\
r_2 & 0 & -r_2 & 0 \\
r_4 & 0 & 0 & -r_4
\end{array}
\]

\[R = \min(r_1, r_3)\]
Outline

1. Introduction
   - Collective Systems
   - Quantitative Analysis
   - Stochastic Process Algebra

2. Quantitative Analysis of Collective Systems
   - Model construction
   - Mathematical analysis: fluid approximation
   - Numerical illustration
   - Deriving properties: fluid model checking

3. Conclusions
Modelling collective behaviour

A key feature of collective systems is the existence of populations of entities who share certain characteristics.
Modelling collective behaviour

A key feature of collective systems is the existence of populations of entities who share certain characteristics.

High-level modelling formalisms allow this repetition to be captured at the high-level rather than explicitly.
Process Algebra for Collective Systems

Process algebra are well-suited for constructing models of collective systems:

- Developed to represent concurrent behaviour compositionally;
Process Algebra for Collective Systems

Process algebra are well-suited for constructing models of collective systems:

- Developed to represent concurrent behaviour compositionally;
- Represent the interactions between individuals explicitly;
Process Algebra for Collective Systems

Process algebra are well-suited for constructing models of collective systems:

- Developed to represent concurrent behaviour compositionally;
- Represent the interactions between individuals explicitly;
- Populations are readily and rigorously identified;
Process Algebra for Collective Systems

Process algebra are well-suited for constructing models of collective systems:

- Developed to represent concurrent behaviour compositionally;
- Represent the interactions between individuals explicitly;
- Populations are readily and rigorously identified;
- Stochastic extensions allow the dynamics of system behaviour to be captured;
Process Algebra for Collective Systems

Process algebra are well-suited for constructing models of collective systems:

- Developed to represent concurrent behaviour compositionally;
- Represent the interactions between individuals explicitly;
- Populations are readily and rigorously identified;
- Stochastic extensions allow the dynamics of system behaviour to be captured;
- Incorporate formal apparatus for reasoning about the behaviour of systems.
Process Algebra for Collective Systems

Process algebra are well-suited for constructing models of collective systems:

- Developed to represent **concurrent** behaviour compositionally;
- Represent the **interactions** between individuals explicitly;
- **Populations** are readily and rigorously identified;
- Stochastic extensions allow the **dynamics** of system behaviour to be captured;
- Incorporate formal apparatus for **reasoning** about the behaviour of systems.

Recent advances in analysis techniques for process algebras have made it possible to study such systems even when the number of entities and activities become huge.
Solving discrete state models

Under the SOS semantics a SPA model is mapped to a **CTMC** with global states determined by the local states of all the participating components.
Solving discrete state models

Under the SOS semantics a SPA model is mapped to a CTMC with global states determined by the local states of all the participating components.
Solving discrete state models

When the size of the state space is not too large they are amenable to numerical solution (linear algebra) to determine a steady state or transient probability distribution.
When the size of the state space is not too large they are amenable to numerical solution (linear algebra) to determine a steady state or transient probability distribution.

\[ Q = \begin{pmatrix}
q_{1,1} & q_{1,2} & \cdots & q_{1,N} \\
q_{2,1} & q_{2,2} & \cdots & q_{2,N} \\
\vdots & \vdots & \ddots & \vdots \\
q_{N,1} & q_{N,2} & \cdots & q_{N,N}
\end{pmatrix} \]
Solving discrete state models

When the size of the state space is not too large they are amenable to numerical solution (linear algebra) to determine a steady state or transient probability distribution.

\[
Q = \begin{pmatrix}
q_{1,1} & q_{1,2} & \cdots & q_{1,N} \\
q_{2,1} & q_{2,2} & \cdots & q_{2,N} \\
\vdots & \vdots & \ddots & \vdots \\
q_{N,1} & q_{N,2} & \cdots & q_{N,N}
\end{pmatrix}
\]

\[
\pi(t) = (\pi_1(t), \pi_2(t), \ldots, \pi_N(t))
\]

\[
\pi(\infty)Q = 0
\]
Solving discrete state models

Alternatively they may be studied using stochastic simulation. Each run generates a single trajectory through the state space. Many runs are needed in order to obtain average behaviours.
As the size of the state space becomes large it becomes infeasible to carry out numerical solution and extremely time-consuming to conduct stochastic simulation.
Identity and Individuality

Collective systems are constructed from many instances of a set of components.
Identity and Individuality

Collective systems are constructed from many instances of a set of components.

If we cease to distinguish between instances of components we can form an aggregation or counting abstraction to reduce the state space.
Identity and Individuality

Collective systems are constructed from many instances of a set of components.

If we cease to distinguish between instances of components we can form an aggregation or counting abstraction to reduce the state space.
Identity and Individuality

Collective systems are constructed from many instances of a set of components.

If we cease to distinguish between instances of components we can form an aggregation or counting abstraction to reduce the state space.
Identity and Individuality

Collective systems are constructed from many instances of a set of components.

If we cease to distinguish between instances of components we can form an aggregation or counting abstraction to reduce the state space.

We may choose to disregard the identity of components.
Identity and Individuality

Collective systems are constructed from many instances of a set of components.

If we cease to distinguish between instances of components we can form an aggregation or counting abstraction to reduce the state space.

We may choose to disregard the identity of components.

Even better reductions can be achieved when we no longer regard the components as individuals.
Population statistics: emergent behaviour

A shift in perspective allows us to model the interactions between individual components but then only consider the system as a whole as an interaction of populations.
A shift in perspective allows us to model the interactions between individual components but then only consider the system as a whole as an interaction of populations.

This allows us to model much larger systems than previously possible but in making the shift we are no longer able to collect any information about individuals in the system.
A shift in perspective allows us to model the interactions between individual components but then only consider the system as a whole as an interaction of populations.

This allows us to model much larger systems than previously possible but in making the shift we are no longer able to collect any information about individuals in the system.

To characterise the behaviour of a population we calculate the proportion of individuals within the population that are exhibiting certain behaviours rather than tracking individuals directly.
A shift in perspective allows us to model the interactions between individual components but then only consider the system as a whole as an interaction of populations.

This allows us to model much larger systems than previously possible but in making the shift we are no longer able to collect any information about individuals in the system.

To characterise the behaviour of a population we calculate the proportion of individuals within the population that are exhibiting certain behaviours rather than tracking individuals directly.

Furthermore we make a continuous approximation of how the proportions vary over time.
Continuous Approximation

This means shifting to a different mathematical representation, where we no longer keep track of the individual states of each entity.
Continuous Approximation

This means shifting to a different mathematical representation, where we no longer keep track of the individual states of each entity.

As we are focussed instead of proportions within populations we now treat these variables as continuous rather than discrete.
Continuous Approximation

This means shifting to a different mathematical representation, where we no longer keep track of the individual states of each entity.

As we are focussed instead of proportions within populations we now treat these variables as continuous rather than discrete.
Continuous Approximation

This means shifting to a different mathematical representation, where we no longer keep track of the individual states of each entity.

As we are focussed instead of proportions within populations we now treat these variables as continuous rather than discrete.
Continuous Approximation

This means shifting to a different mathematical representation, where we no longer keep track of the individual states of each entity.

As we are focussed instead of proportions within populations we now treat these variables as continuous rather than discrete.
Continuous Approximation

This means shifting to a different mathematical representation, where we no longer keep track of the individual states of each entity.

As we are focussed instead of proportions within populations we now treat these variables as continuous rather than discrete.
Continuous Approximation

This means shifting to a different mathematical representation, where we no longer keep track of the individual states of each entity.

As we are focussed instead of proportions within populations we now treat these variables as continuous rather than discrete.
Continuous Approximation

This means shifting to a different mathematical representation, where we no longer keep track of the individual states of each entity.

As we are focussed instead of proportions within populations we now treat these variables as continuous rather than discrete.
Continuous Approximation

This means shifting to a different mathematical representation, where we no longer keep track of the individual states of each entity.

As we are focused instead of proportions within populations we now treat these variables as continuous rather than discrete.
Continuous Approximation

This means shifting to a different mathematical representation, where we no longer keep track of the individual states of each entity.

As we are focussed instead of proportions within populations we now treat these variables as continuous rather than discrete.
Continuous Approximation

This means shifting to a different mathematical representation, where we no longer keep track of the individual states of each entity.

As we are focussed instead of *proportions within populations* we now treat these variables as *continuous* rather than *discrete*.
Continuous Approximation

This means shifting to a different mathematical representation, where we no longer keep track of the individual states of each entity.

As we are focussed instead of proportions within populations we now treat these variables as continuous rather than discrete.
Continuous Approximation

This means shifting to a different mathematical representation, where we no longer keep track of the individual states of each entity.

As we are focussed instead of proportions within populations we now treat these variables as continuous rather than discrete.
Continuous Approximation

This means shifting to a different mathematical representation, where we no longer keep track of the individual states of each entity.

As we are focused instead of proportions within populations, we now treat these variables as continuous rather than discrete.

Use ordinary differential equations to represent the evolution of those variables over time.
Simple example revisited

\[
\begin{align*}
\text{Proc}_0 & \overset{\text{def}}{=} (\text{task}1, r_1).\text{Proc}_1 \\
\text{Proc}_1 & \overset{\text{def}}{=} (\text{task}2, r_2).\text{Proc}_0 \\
\text{Res}_0 & \overset{\text{def}}{=} (\text{task}1, r_3).\text{Res}_1 \\
\text{Res}_1 & \overset{\text{def}}{=} (\text{reset}, r_4).\text{Res}_0
\end{align*}
\]

\[\text{Proc}_0[N_P] \boxtimes_{\{\text{task}1\}} \text{Res}_0[N_R]\]
Simple example revisited

\[ \text{Proc}_0 \overset{\text{def}}{=} (\text{task1}, r_1).\text{Proc}_1 \]
\[ \text{Proc}_1 \overset{\text{def}}{=} (\text{task2}, r_2).\text{Proc}_0 \]
\[ \text{Res}_0 \overset{\text{def}}{=} (\text{task1}, r_3).\text{Res}_1 \]
\[ \text{Res}_1 \overset{\text{def}}{=} (\text{reset}, r_4).\text{Res}_0 \]

CTMC interpretation

<table>
<thead>
<tr>
<th>Processors ((N_P))</th>
<th>Resources ((N_R))</th>
<th>States ((2^{N_P + N_R}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>128</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>256</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>512</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1024</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>2048</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>4096</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>8192</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>16384</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>32768</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>65536</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>131072</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>262144</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>524288</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>1048576</td>
</tr>
</tbody>
</table>
Simple example revisited

\[
\begin{align*}
Proc_0 & \overset{\text{def}}{=} (\text{task1}, r_1).Proc_1 \\
Proc_1 & \overset{\text{def}}{=} (\text{task2}, r_2).Proc_0 \\
Res_0 & \overset{\text{def}}{=} (\text{task1}, r_3).Res_1 \\
Res_1 & \overset{\text{def}}{=} (\text{reset}, r_4).Res_0 \\
Proc_0[N_P] & \otimes_{\{\text{task1}\}} Res_0[N_R]
\end{align*}
\]

- \text{task1} decreases \(Proc_0\) and \(Res_0\)
- \text{task1} increases \(Proc_1\) and \(Res_1\)
- \text{task2} decreases \(Proc_1\)
- \text{task2} increases \(Proc_0\)
- \text{reset} decreases \(Res_1\)
- \text{reset} increases \(Res_0\)
Simple example revisited

\[ \frac{dx_1}{dt} = -\min(r_1 x_1, r_3 x_3) + r_2 x_2 \]
\[ x_1 = \text{no. of } \text{Proc}_1 \]

- \text{task1 decreases } \text{Proc}_0
- \text{task1 is performed by } \text{Proc}_0 \text{ and } \text{Res}_0
- \text{task2 increases } \text{Proc}_0
- \text{task2 is performed by } \text{Proc}_1
Simple example revisited

\[ Proc_0 = (\text{task1}, r_1).Proc_1 \]
\[ Proc_1 = (\text{task2}, r_2).Proc_0 \]
\[ Res_0 = (\text{task1}, r_3).Res_1 \]
\[ Res_1 = (\text{reset}, r_4).Res_0 \]

\[ Proc_0[N_P] \preceq Res_0[N_R] \]

**ODE interpretation**

\[ \frac{dx_1}{dt} = -\min(r_1 x_1, r_3 x_3) + r_2 x_2 \]
\[ x_1 = \text{no. of } Proc_1 \]

\[ \frac{dx_2}{dt} = \min(r_1 x_1, r_3 x_3) - r_2 x_2 \]
\[ x_2 = \text{no. of } Proc_2 \]

\[ \frac{dx_3}{dt} = -\min(r_1 x_1, r_3 x_3) + r_4 x_4 \]
\[ x_3 = \text{no. of } Res_0 \]

\[ \frac{dx_4}{dt} = \min(r_1 x_1, r_3 x_3) - r_4 x_4 \]
\[ x_4 = \text{no. of } Res_1 \]
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 (ODE solution)
Deriving a Fluid Approximation of a SPA model

The aim is to represent the CTMC implicitly (avoiding state space explosion), and to generate the set of ODEs which are the fluid limit of that CTMC.
Deriving a Fluid Approximation of a SPA model

The aim is to represent the CTMC implicitly (avoiding state space explosion), and to generate the set of ODEs which are the fluid limit of that CTMC.

The existing (CTMC) SOS semantics is not suitable for this purpose because it constructs the state space of the CTMC explicitly.
Deriving a Fluid Approximation of a SPA model

The aim is to represent the CTMC implicitly (avoiding state space explosion), and to generate the set of ODEs which are the fluid limit of that CTMC.

The existing (CTMC) SOS semantics is not suitable for this purpose because it constructs the state space of the CTMC explicitly.

\[ \text{SPA MODEL} \xrightarrow{\text{SOS rules}} \text{LABELLED TRANSITION SYSTEM} \xrightarrow{\text{state transition diagram}} \text{CTMC Q} \]
Deriving a Fluid Approximation of a SPA model

The aim is to represent the CTMC implicitly (avoiding state space explosion), and to generate the set of ODEs which are the fluid limit of that CTMC.

The existing (CTMC) SOS semantics is not suitable for this purpose because it constructs the state space of the CTMC explicitly.

We define a structured operational semantics which defines the possible transitions of an arbitrary abstract state and from this derive the ODEs.
Deriving a Fluid Approximation of a SPA model

The aim is to represent the CTMC implicitly (avoiding state space explosion), and to generate the set of ODEs which are the fluid limit of that CTMC.

The existing (CTMC) SOS semantics is not suitable for this purpose because it constructs the state space of the CTMC explicitly.

We define a structured operational semantics which defines the possible transitions of an arbitrary abstract state and from this derive the ODEs.

\[ \text{SPA MODEL} \xrightarrow{\text{SOS rules}} \text{SYMBOLIC LABELLED TRANSITION SYSTEM} \xrightarrow{\text{generator functions}} \text{ABSTRACT CTMC } Q \text{ or ODEs } F_M(x) \]
Fluid Structured Operational Semantics

In order to get to the implicit representation of the CTMC we need to:

1. Remove excess components to identify the counting abstraction of the process (Context Reduction)
2. Collect the transitions of the reduced context as symbolic updates on the state representation (Jump Multiset)
3. Calculate the rate of the transitions in terms of an arbitrary state of the CTMC.

Once this is done we can extract the vector field $F_M(x)$ from the jump multiset, under the assumption that the population size tends to infinity.

Fluid Structured Operational Semantics

In order to get to the implicit representation of the CTMC we need to:

1. Remove excess components to identify the counting abstraction of the process (Context Reduction)
Fluid Structured Operational Semantics

In order to get to the implicit representation of the CTMC we need to:

1. Remove excess components to identify the counting abstraction of the process (Context Reduction)
2. Collect the transitions of the reduced context as symbolic updates on the state representation (Jump Multiset)
Fluid Structured Operational Semantics

In order to get to the implicit representation of the CTMC we need to:

1. Remove excess components to identify the counting abstraction of the process (Context Reduction)

2. Collect the transitions of the reduced context as symbolic updates on the state representation (Jump Multiset)

3. Calculate the rate of the transitions in terms of an arbitrary state of the CTMC.
Fluid Structured Operational Semantics

In order to get to the implicit representation of the CTMC we need to:

1. Remove excess components to identify the counting abstraction of the process (Context Reduction)

2. Collect the transitions of the reduced context as symbolic updates on the state representation (Jump Multiset)

3. Calculate the rate of the transitions in terms of an arbitrary state of the CTMC.

Once this is done we can extract the vector field $F_M(x)$ from the jump multiset, under the assumption that the population size tends to infinity.

Context Reduction

\[
\begin{align*}
\text{Proc}_0 & \overset{\text{def}}{=} (\text{task}1, r_1).\text{Proc}_1 \\
\text{Proc}_1 & \overset{\text{def}}{=} (\text{task}2, r_2).\text{Proc}_0 \\
\text{Res}_0 & \overset{\text{def}}{=} (\text{task}1, r_3).\text{Res}_1 \\
\text{Res}_1 & \overset{\text{def}}{=} (\reset, r_4).\text{Res}_0 \\
\text{System} & \overset{\text{def}}{=} \text{Proc}_0[N_P] \otimes \text{Res}_0[N_R] \\
\downarrow \\
\mathcal{R}(\text{System}) & = \{\text{Proc}_0, \text{Proc}_1\} \otimes_{\{\text{task}1\}} \{\text{Res}_0, \text{Res}_1\}
\end{align*}
\]
Context Reduction

\[ \begin{align*}
    \text{Proc}_0 & \overset{\text{def}}{=} (\text{task}1, r_1).\text{Proc}_1 \\
    \text{Proc}_1 & \overset{\text{def}}{=} (\text{task}2, r_2).\text{Proc}_0 \\
    \text{Res}_0 & \overset{\text{def}}{=} (\text{task}1, r_3).\text{Res}_1 \\
    \text{Res}_1 & \overset{\text{def}}{=} (\text{reset}, r_4).\text{Res}_0 \\
    \text{System} & \overset{\text{def}}{=} \text{Proc}_0[N_P] \{\text{transfer}\} \text{Res}_0[N_R] \\
\end{align*} \]

\[ \Downarrow \]

\[ \mathcal{R}(\text{System}) = \{\text{Proc}_0, \text{Proc}_1\} \{\text{task}1\} \{\text{Res}_0, \text{Res}_1\} \]

Population Vector

\[ \xi = (\xi_1, \xi_2, \xi_3, \xi_4) \]
Location Dependency

\[
System \overset{\text{def}}{=} \text{Proc}_0[N'_C] \parallel \{\text{task1}\} \text{Res}_0[N_S] \parallel \text{Proc}_0[N''_C]
\]
Location Dependency

\[ \text{System} \overset{\text{def}}{=} \text{Proc}_0[N'_C] \parallel \text{Res}_0[N_S] \parallel \text{Proc}_0[N''_C] \]

\[ \downarrow \]

\[ \{\text{Proc}_0, \text{Proc}_1\} \parallel \{\text{Res}_0, \text{Res}_1\} \parallel \{\text{Proc}_0, \text{Proc}_1\} \]
**Location Dependency**

\[
\text{System} \overset{\text{def}}{=} \text{Proc}_0[N'_C] \uparrow \{ \text{task1} \} \text{Res}_0[N_S] \parallel \text{Proc}_0[N''_C]
\]

\[
\downarrow
\]

\[
\{ \text{Proc}_0, \text{Proc}_1 \} \uparrow \{ \text{Res}_0, \text{Res}_1 \} \parallel \{ \text{Proc}_0, \text{Proc}_1 \}
\]

**Population Vector**

\[
\xi = (\xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \xi_6)
\]
Fluid Structured Operational Semantics by Example

\[
\begin{align*}
Proc_0 & \overset{\text{def}}{=} (\text{task}1, r_1).Proc_1 \\
Proc_1 & \overset{\text{def}}{=} (\text{task}2, r_2).Proc_0 \\
Res_0 & \overset{\text{def}}{=} (\text{task}1, r_3).Res_1 \\
Res_1 & \overset{\text{def}}{=} (\text{reset}, r_4).Res_0 \\
\text{System} & \overset{\text{def}}{=} Proc_0[N_P] \{\text{transfer}\} Res_0[N_R] \\
\xi & = (\xi_1, \xi_2, \xi_3, \xi_4)
\end{align*}
\]
Fluid Structured Operational Semantics by Example

\[
\begin{align*}
\text{Proc}_0 & \overset{\text{def}}{=} (\text{task}1, r_1).\text{Proc}_1 \\
\text{Proc}_1 & \overset{\text{def}}{=} (\text{task}2, r_2).\text{Proc}_0 \\
\text{Res}_0 & \overset{\text{def}}{=} (\text{task}1, r_3).\text{Res}_1 \\
\text{Res}_1 & \overset{\text{def}}{=} (\text{reset}, r_4).\text{Res}_0 \\
\text{System} & \overset{\text{def}}{=} \text{Proc}_0[N_P] \otimes_{\text{transfer}} \text{Res}_0[N_R] \\
\xi &= (\xi_1, \xi_2, \xi_3, \xi_4)
\end{align*}
\]

\[
\begin{array}{c}
\text{Proc}_0 \xrightarrow{\text{task}1, r_1} \text{Proc}_1 \\
\text{Proc}_0 \xrightarrow{\text{task}1, r_1 \xi_1} \ast \text{Proc}_1
\end{array}
\]
Fluid Structured Operational Semantics by Example

\[
\begin{align*}
\text{Proc}_0 & \overset{\text{def}}{=} (\text{task}1, r_1) \cdot \text{Proc}_1 \\
\text{Proc}_1 & \overset{\text{def}}{=} (\text{task}2, r_2) \cdot \text{Proc}_0 \\
\text{Res}_0 & \overset{\text{def}}{=} (\text{task}1, r_3) \cdot \text{Res}_1 \\
\text{Res}_1 & \overset{\text{def}}{=} (\text{reset}, r_4) \cdot \text{Res}_0 \\
\text{System} & \overset{\text{def}}{=} \text{Proc}_0[N_P] \{\text{transfer}\} \text{Res}_0[N_R] \\
\xi &= (\xi_1, \xi_2, \xi_3, \xi_4)
\end{align*}
\]

\[
\begin{align*}
\text{Proc}_0 & \xrightarrow{\text{task}1, r_1} \text{Proc}_1 \\
\text{Proc}_0 & \xrightarrow{\text{task}1, r_1 \xi_1} \ast \text{Proc}_1 \\
\text{Res}_0 & \xrightarrow{\text{task}1, r_3} \text{Res}_1 \\
\text{Res}_0 & \xrightarrow{\text{task}1, r_3 \xi_3} \ast \text{Res}_1
\end{align*}
\]
Fluid Structured Operational Semantics by Example

\[\text{Proc}_0 \overset{\text{def}}{=} (\text{task}1, r_1).\text{Proc}_1\]
\[\text{Proc}_1 \overset{\text{def}}{=} (\text{task}2, r_2).\text{Proc}_0\]
\[\text{Res}_0 \overset{\text{def}}{=} (\text{task}1, r_3).\text{Res}_1\]
\[\text{Res}_1 \overset{\text{def}}{=} (\text{reset}, r_4).\text{Res}_0\]
\[\text{System} \overset{\text{def}}{=} \text{Proc}_0[N_P] \overset{\text{transfer}}{\bowtie} \text{Res}_0[N_R]\]

\[\xi = (\xi_1, \xi_2, \xi_3, \xi_4)\]
Apparent Rate Calculation

\[
\begin{align*}
\text{Proc}_0 \xrightarrow{\text{task1}, r_1} & \text{Proc}_1 \\
\text{Proc}_0 \xrightarrow{\text{task1}, r_1 \xi_1} & * \text{Proc}_1 \\
\text{Proc}_0 \xleftarrow{\text{task1}, r_1 \xi_1} & * \text{Proc}_1 \\
\text{Res}_0 \xrightarrow{\text{task1}, r_3} & \text{Res}_1 \\
\text{Res}_0 \xrightarrow{\text{task1}, r_3 \xi_3} & * \text{Res}_1 \\
\text{Proc}_0 \xleftarrow{\{\text{task1}\}} & \text{Res}_0 \xrightarrow{\text{task1}, r(\xi)} * \text{Proc}_1 \xleftarrow{\{\text{task1}\}} \text{Res}_1
\end{align*}
\]
Apparent Rate Calculation

\[
\begin{align*}
\text{Proc}_0 & \xrightarrow{\text{task1}, r_1} \text{Proc}_1 \\
\text{Proc}_0 & \xrightarrow{\text{task1}, r_1 \xi_1} \ast \text{Proc}_1 \\
\text{Proc}_0 & \bigcirc \text{ Res}_0 \xrightarrow{\text{task1}, r(\xi)} \ast \text{Proc}_1 \\
\text{Res}_0 & \xrightarrow{\text{task1}, r_3} \text{Res}_1 \\
\text{Res}_0 & \xrightarrow{\text{task1}, r_3 \xi_3} \ast \text{Res}_1
\end{align*}
\]

\[
r(\xi) = \min (r_1 \xi_1, r_3 \xi_4)
\]
$f(\xi, l, \alpha)$ as the Generator Matrix of the *Lumped* CTMC

\[
\begin{align*}
(P_0 \parallel P_0 \parallel P_0) \quad \{\text{task1}\} \quad (R_0 \parallel R_0) \\
(P_1 \parallel P_0 \parallel P_0) \quad \{\text{task1}\} \quad (R_0 \parallel R_1) \\
(P_0 \parallel P_1 \parallel P_0) \quad \{\text{task1}\} \quad (R_1 \parallel R_0) \\
(P_0 \parallel P_0 \parallel P_1) \quad \{\text{task1}\} \quad (R_1 \parallel R_0) \\
(P_0 \parallel P_0 \parallel P_1) \quad \{\text{task1}\} \quad (R_0 \parallel R_1)
\end{align*}
\]
$f(\xi, l, \alpha)$ as the Generator Matrix of the \textit{Lumped} CTMC

\[
(3, 0, 2, 0) \xrightarrow{\min(3r_1, 2r_3)} (2, 1, 1, 1)
\]

\[
(P_0 \parallel P_0 \parallel P_0) \xrightarrow{\{\text{task1}\}} (R_0 \parallel R_0)
\]

\[
(P_1 \parallel P_0 \parallel P_0) \xrightarrow{\{\text{task1}\}} (R_1 \parallel R_0)
\]

\[
(P_0 \parallel P_1 \parallel P_0) \xrightarrow{\{\text{task1}\}} (R_1 \parallel R_0)
\]

\[
(P_0 \parallel P_1 \parallel P_0) \xrightarrow{\{\text{task1}\}} (R_1 \parallel R_0)
\]

\[
(P_0 \parallel P_1 \parallel P_0) \xrightarrow{\{\text{task1}\}} (R_1 \parallel R_0)
\]

\[
(P_0 \parallel P_0 \parallel P_1) \xrightarrow{\{\text{task1}\}} (R_1 \parallel R_0)
\]

\[
(P_0 \parallel P_0 \parallel P_1) \xrightarrow{\{\text{task1}\}} (R_1 \parallel R_0)
\]

\[
(P_0 \parallel P_0 \parallel P_1) \xrightarrow{\{\text{task1}\}} (R_0 \parallel R_1)
\]

\[
(P_0 \parallel P_0 \parallel P_1) \xrightarrow{\{\text{task1}\}} (R_0 \parallel R_1)
\]
Jump Multiset

\[ Proc_0 \begin{array}{c} \uparrow \\ \{ \text{task1} \} \end{array} Res_0 \xrightarrow{\text{task1, } r(\xi)} * Proc_1 \begin{array}{c} \uparrow \\ \{ \text{task1} \} \end{array} Res_1 \]

\[ r(\xi) = \min (r_1 \xi_1, r_3 \xi_3) \]
Jump Multiset

\[ Proc_0 \{ \text{task1} \} \rightarrow_{\text{task1}, r(\xi)}^* Proc_1 \{ \text{task1} \} \]

\[ r(\xi) = \min (r_1 \xi_1, r_3 \xi_3) \]

\[ Proc_1 \{ \text{task1} \} \rightarrow_{\text{task2}, \xi_2 r_2}^* Proc_0 \{ \text{task1} \} \rightarrow_{\text{reset}, \xi_4 r_4}^* Proc_0 \{ \text{task1} \} \]
Jump Multiset

\[ Proc_0 \quad \{ \text{task1} \} \quad Res_0 \xrightarrow{\text{task1}, r(\xi)} \quad \ast \quad Proc_1 \quad \{ \text{task1} \} \quad Res_1 \]

\[ r(\xi) = \min \left( r_1 \xi_1, r_3 \xi_3 \right) \]

\[ Proc_1 \quad \{ \text{task1} \} \quad Res_0 \xrightarrow{\text{task2}, \xi_2 r_2} \quad \ast \quad Proc_0 \quad \{ \text{task1} \} \quad Res_0 \]

\[ Proc_0 \quad \{ \text{task1} \} \quad Res_1 \xrightarrow{\text{reset}, \xi_4 r_4} \quad \ast \quad Proc_0 \quad \{ \text{task1} \} \quad Res_0 \]
Some transitions may give the same information:

\[ \text{Proc}_0 \left\{ \{\text{task1}\} \right\} \rightarrow \text{Res}_1 \xrightarrow{\text{reset, } \xi_4 r_4} \star \rightarrow \text{Proc}_0 \left\{ \{\text{task1}\} \right\} \]

\[ \text{Proc}_1 \left\{ \{\text{task1}\} \right\} \rightarrow \text{Res}_1 \xrightarrow{\text{reset, } \xi_4 r_4} \star \rightarrow \text{Proc}_1 \left\{ \{\text{task1}\} \right\} \]

i.e., \(\text{Res}_1\) may perform an action independently from the rest of the system.

This is captured by the procedure used for the construction of the generator function \(f(\xi, l, \alpha)\)
Construction of $f(\xi, l, \alpha)$

$Proc_0 \begin{array}{c} \begin{array}{c} \text{reset, } \xi_4 r_4 \end{array} \end{array} \quad Res_1 \rightarrow \star \quad Proc_0 \begin{array}{c} \begin{array}{c} \text{task1} \end{array} \end{array} \quad Res_0$

$Take l = (0, 0, 0, 0)$

$Add -1 to all elements of l corresponding to the indices of the components in the lhs of the transition$

$l = (-1, 0, 0, -1)$

$Add +1 to all elements of l corresponding to the indices of the components in the rhs of the transition$

$l = (0, 0, +1, -1) = (0, 0, +1, -1)$

$f(\xi, (0, 0, +1, -1), \text{reset}) = \xi_4 r_4$
Construction of $f(\xi, I, \alpha)$

Take $I = (0, 0, 0, 0)$
Construction of $f(\xi, l, \alpha)$

- Take $l = (0, 0, 0, 0)$
- Add $-1$ to all elements of $l$ corresponding to the indices of the components in the lhs of the transition.
  
  $l = (-1, 0, 0, -1)$

- Add $+1$ to all elements of $l$ corresponding to the indices of the components in the rhs of the transition.
  
  $l = (-1 + 1, 0, +1, -1) = (0, 0, +1, -1)$
Construction of $f(\xi, l, \alpha)$

- Take $l = (0, 0, 0, 0)$
- Add $-1$ to all elements of $l$ corresponding to the indices of the components in the lhs of the transition

$$l = (-1, 0, 0, -1)$$

- Add $+1$ to all elements of $l$ corresponding to the indices of the components in the rhs of the transition

$$l = (-1 + 1, 0, +1, -1) = (0, 0, +1, -1)$$

$$f(\xi, (0, 0, +1, -1), \text{reset}) = \xi_4 r_4$$
Construction of $f(\xi, l, \alpha)$
Construction of $f(\xi, l, \alpha)$

$$f(\xi, (-1, +1, -1, +1), task1) = r(\xi)$$
Construction of $f(\xi, l, \alpha)$

\[
\begin{align*}
\text{Proc}_0 \quad \begin{array}{c} \text{Res}_0 \\ \{\text{task1}\} \end{array} & \xrightarrow{\text{task1, } r(\xi)} \begin{array}{c} \text{Res}_1 \\ \{\text{task1}\} \end{array} \quad \text{Proc}_1 \\
\text{Proc}_1 \quad \begin{array}{c} \text{Res}_0 \\ \{\text{task1}\} \end{array} & \xrightarrow{\text{task2, } \xi_2 r_2'} \begin{array}{c} \text{Res}_0 \\ \{\text{task1}\} \end{array} \quad \text{Proc}_0 \\

f(\xi, (-1, +1, -1, +1), \text{task1}) & = r(\xi) \\
f(\xi, (+1, -1, 0, 0), \text{task2}) & = \xi_2 r_2
\end{align*}
\]
Construction of $f(\xi, l, \alpha)$

\begin{align*}
Proc_0 \{\text{task1}\} & \rightarrow \quad Res_0 & \quad \xrightarrow{\text{task1, } r(\xi)} & \quad Proc_1 \{\text{task1}\} \\
Proc_1 \{\text{task1}\} & \rightarrow \quad Res_0 & \quad \xrightarrow{\text{task2, } \xi_2 r'_2} & \quad Proc_0 \{\text{task1}\} \\
Proc_0 \{\text{task1}\} & \rightarrow \quad Res_1 & \quad \xrightarrow{\text{reset, } \xi_4 r_4} & \quad Proc_0 \{\text{task1}\} \\

f(\xi, (-1, +1, -1, +1), \text{task1}) & = \quad r(\xi) \\
f(\xi, (+1, -1, 0, 0), \text{task2}) & = \quad \xi_2 r_2 \\
f(\xi, (0, 0, +1, -1), \text{reset}) & = \quad \xi_4 r_4
\end{align*}
Capturing behaviour in the Generator Function

\[
\begin{align*}
Proc_0 & \overset{\text{def}}{=} (task_1, r_1).Proc_1 \\
Proc_1 & \overset{\text{def}}{=} (task_2, r_2).Proc_0 \\
Res_0 & \overset{\text{def}}{=} (task_1, r_3).Res_1 \\
Res_1 & \overset{\text{def}}{=} (reset, r_4).Res_0 \\
System & \overset{\text{def}}{=} Proc_0[N_P] \ {\text{transfer}} \ Res_0[N_R]
\end{align*}
\]
Capturing behaviour in the Generator Function

\[\begin{align*}
Proc_0 & \overset{\text{def}}{=} (\text{task1}, r_1).Proc_1 \\
Proc_1 & \overset{\text{def}}{=} (\text{task2}, r_2).Proc_0 \\
Res_0 & \overset{\text{def}}{=} (\text{task1}, r_3).Res_1 \\
Res_1 & \overset{\text{def}}{=} (\text{reset}, r_4).Res_0 \\
\text{System} & \overset{\text{def}}{=} Proc_0[N_P] \bigcirc \left\{ \text{transfer} \right\} Res_0[N_R]
\end{align*}\]

Numerical Vector Form

\[\xi = (\xi_1, \xi_2, \xi_3, \xi_4) \in \mathbb{N}^4, \quad \xi_1 + \xi_2 = N_P \quad \text{and} \quad \xi_3 + \xi_4 = N_R\]
Capturing behaviour in the Generator Function

\[
\begin{align*}
Proc_0 & \overset{\text{def}}{=} (\text{task1}, r_1).Proc_1 \\
Proc_1 & \overset{\text{def}}{=} (\text{task2}, r_2).Proc_0 \\
Res_0 & \overset{\text{def}}{=} (\text{task1}, r_3).Res_1 \\
Res_1 & \overset{\text{def}}{=} (\text{reset}, r_4).Res_0 \\
System & \overset{\text{def}}{=} Proc_0[N_P] \bigoplus \{\text{transfer}\} \ Res_0[N_R]
\end{align*}
\]

Numerical Vector Form

\[\xi = (\xi_1, \xi_2, \xi_3, \xi_4) \in \mathbb{N}^4, \quad \xi_1 + \xi_2 = N_P \quad \text{and} \quad \xi_3 + \xi_4 = N_R\]

Generator Function

\[
\begin{align*}
f(\xi, (-1, 1, -1, 1), \text{task1}) & = \min (r_1 \xi_1, r_3 \xi_3) \\
f(\xi, (1, -1, 0, 0), \text{task2}) & = r_2 \xi_2 \\
f(\xi, (0, 0, 1, -1), \text{reset}) & = r_4 \xi_4
\end{align*}
\]
Extraction of the ODE from \( f \)

**Generator Function**

\[
\begin{align*}
    f(\xi, (-1, 1, -1, 1), \text{task1}) & = \min (r_1\xi_1, r_3\xi_3) \\
    f(\xi, (1, -1, 0, 0), \text{task2}) & = r_2\xi_2 \\
    f(\xi, (0, 0, 1, -1), \text{reset}) & = r_4\xi_4
\end{align*}
\]

**Differential Equation**

\[
\frac{dx}{dt} = F_M(x) = \sum_{l \in \mathbb{Z}^d} l \sum_{\alpha \in A} f(x, l, \alpha)
\]

\[
= (-1, 1, -1, 1) \min (r_1x_1, r_3x_3) + (1, -1, 0, 0)r_2x_2 \\
+ (0, 0, 1, -1)r_4x_4
\]
Extraction of the ODE from $f$

**Generator Function**

$$f(\xi, (-1, 1, -1, 1), task1) = \min (r_1 \xi_1, r_3 \xi_3)$$
$$f(\xi, (1, -1, 0, 0), task2) = r_2 \xi_2$$
$$f(\xi, (0, 0, 1, -1), reset) = r_4 \xi_4$$

**Differential Equation**

$$\frac{dx_1}{dt} = - \min (r_1 x_1, r_3 x_3) + r_2 x_2$$
$$\frac{dx_2}{dt} = \min (r_1 x_1, r_3 x_3) - r_2 x_2$$
$$\frac{dx_3}{dt} = - \min (r_1 x_1, r_3 x_3) + r_4 x_4$$
$$\frac{dx_4}{dt} = \min (r_1 x_1, r_3 x_3) - r_4 x_4$$
Consistency results

- The vector field $\mathcal{F}(x)$ is Lipschitz continuous i.e. all the rate functions governing transitions in the process algebra satisfy local continuity conditions.
Consistency results

- The vector field $\mathcal{F}(x)$ is Lipschitz continuous i.e. all the rate functions governing transitions in the process algebra satisfy local continuity conditions.

- The generated ODEs are the fluid limit of the family of CTMCs: this family forms a sequence as the initial populations are scaled by a variable $n$. 

---

Consistency results

- The vector field $F(x)$ is Lipschitz continuous i.e. all the rate functions governing transitions in the process algebra satisfy local continuity conditions.

- The generated ODEs are the fluid limit of the family of CTMCs: this family forms a sequence as the initial populations are scaled by a variable $n$.

- We can prove this using Kurtz’s theorem:
  
Consistency results

- The vector field $\mathcal{F}(x)$ is Lipschitz continuous i.e. all the rate functions governing transitions in the process algebra satisfy local continuity conditions.

- The generated ODEs are the fluid limit of the family of CTMCs: this family forms a sequence as the initial populations are scaled by a variable $n$.


- Moreover Lipschitz continuity of the vector field guarantees existence and uniqueness of the solution to the initial value problem.

Quantitative properties

The derived vector field $\mathcal{F}(x)$, gives an approximation of the expected count for each population over time.
Quantitative properties

The derived vector field $\mathcal{F}(x)$, gives an approximation of the expected count for each population over time.

This has been extended in a number of ways:

- **Fluid rewards** which can be safely calculated from the fluid expectation trajectories.

Quantitative properties

The derived vector field $\mathcal{F}(x)$, gives an approximation of the expected count for each population over time.

This has been extended in a number of ways:

- **Fluid rewards** which can be safely calculated from the fluid expectation trajectories.


- **Vector fields** have been defined to approximate higher moments.

Quantitative properties

The derived vector field $\mathcal{F}(x)$, gives an approximation of the expected count for each population over time.

This has been extended in a number of ways:

- **Fluid rewards** which can be safely calculated from the fluid expectation trajectories.
  

- **Vector fields** have been defined to approximate higher moments.
  

- **Fluid approximation of passage times** have been defined.
  
Fluid model checking

Since the vector field records only deterministic behaviour, LTL model checking can be used over a trace to give boolean results.
Since the vector field records only deterministic behaviour, LTL model checking can be used over a trace to give boolean results.

But for the systems we are interested in we would like some more quantified answers, in the style of stochastic model checking.
Fluid model checking

Since the vector field records only deterministic behaviour, LTL model checking can be used over a trace to give boolean results.

But for the systems we are interested in we would like some more quantified answers, in the style of stochastic model checking.

Work on this is on-going but there are initial results for:

- **CSL properties of a single agent** within a population.
  

- **The fraction of a population** that satisfies a property expressed as a one-clock deterministic timed automaton.
  
We consider properties of a single agent within a population, expressed in the Continuous Stochastic Logic (CSL), usually used for model checking CTMCs.
CSL model checking of a single agent

We consider properties of a single agent within a population, expressed in the Continuous Stochastic Logic (CSL), usually used for model checking CTMCs.

We consider an arbitrary member of the population.
CSL model checking of a single agent

We consider properties of a single agent within a population, expressed in the Continuous Stochastic Logic (CSL), usually used for model checking CTMCs.

This agent is kept discrete, making transitions between its discrete states, but all other agents are treated as a mean-field influencing the behaviour of this agent.
CSL model checking of a single agent

We consider properties of a single agent within a population, expressed in the Continuous Stochastic Logic (CSL), usually used for model checking CTMCs.

Essentially we keep a detailed discrete-event representation of the one agent and make a fluid approximation of the rest of the population.
Inhomogeneous CTMC

The transition rates within the discrete-event representation will depend on the rest of the population.

i.e. it will depend on the vector field capturing the behaviour of the residual population.

\[ \begin{align*}
    r_1 &\quad r_3 \\
    r_2 &\quad r_4 \\
    f_1 &\quad f_3 \\
    f_2 &\quad f_4
\end{align*} \]

where \( f_i = f(\text{[vector field]}) \)
Inhomogeneous CTMC

The transition rates within the discrete-event representation will depend on the rest of the population.

i.e. it will depend on the vector field capturing the behaviour of the residual population.

\[ r_1, r_3 \Rightarrow f_1, f_3 \]

where \( f_i = f(\cdot) \)

It is an inhomogeneous continuous time Markov chain.
Model checking the ICTMC

Care is needed to model check the ICTMC, which proceeds by explicitly calculating the reachability probability probabilities for states of interest (analogously to CSL model checking on CTMCs).
Model checking the ICTMC

Care is needed to model check the ICTMC, which proceeds by explicitly calculating the reachability probability probabilities for states of interest (analogously to CSL model checking on CTMCs).

The inhomogeneous time within the model means that truth values may change with respect to time.
Outline

1 Introduction
   • Collective Systems
   • Quantitative Analysis
   • Stochastic Process Algebra

2 Quantitative Analysis of Collective Systems
   • Model construction
   • Mathematical analysis: fluid approximation
   • Numerical illustration
   • Deriving properties: fluid model checking

3 Conclusions
Collective Systems are an interesting and challenging class of systems to design and construct.
Collective Systems are an interesting and challenging class of systems to design and construct.

Their role within infrastructure, such as within smart cities, make it essential that quantitative aspects of behaviour is taken into consideration, as well as functional correctness.
Conclusions

■ Collective Systems are an interesting and challenging class of systems to design and construct.

■ Their role within infrastructure, such as within smart cities, make it essential that quantitative aspects of behaviour is taken into consideration, as well as functional correctness.

■ Fluid approximation based analysis offers hope for scalable quantitative analysis techniques, but there remain many interesting and challenging problems to be solved.
Collective Systems are an interesting and challenging class of systems to design and construct.

Their role within infrastructure, such as within smart cities, make it essential that quantitative aspects of behaviour is taken into consideration, as well as functional correctness.

Fluid approximation based analysis offers hope for scalable quantitative analysis techniques, but there remain many interesting and challenging problems to be solved.

In particular we currently seek to bring the fluid approximation techniques to systems with distinct locations.
Thank you!

[www.quanticol.eu]
Thank you!

Thanks to the other members of the QUANTICOL project

quanticol

www.quanticol.eu