Integrating Inference with Stochastic Process Algebra Models

Jane Hillston

School of Informatics, University of Edinburgh

11th January 2018
Outline

1. Stochastic Process Algebras
2. Modelling in a Data Rich World
3. ProPPA
4. Inference
5. Results
6. Conclusions
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.

\[ \alpha.P \]

- 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 algebras where models are decorated with quantitative information used to generate a stochastic process are stochastic process algebras (SPA).
Stochastic Process Algebra

- Models are constructed from components which engage in activities.

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

- The language is used to generate a Continuous Time Markov Chain (CTMC) for performance modelling.

- The CTMC can be analysed numerically (linear algebra) or by stochastic simulation.
Stochastic Process Algebra

- Models are constructed from components which engage in activities.

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

- The language is used to generate a Continuous Time Markov Chain (CTMC) for performance modelling.

- The CTMC can be analysed numerically (linear algebra) or by stochastic simulation.
Stochastic Process Algebra

- Models are constructed from components which engage in activities.

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

- The language is used to generate a Continuous Time Markov Chain (CTMC) for performance modelling.

- The CTMC can be analysed numerically (linear algebra) or by stochastic simulation.
Integrated analysis

Qualitative verification complemented by quantitative verification.
Integrated analysis

**Qualitative** verification complemented by **quantitative** verification.

Reachability analysis

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

Qualitative verification complemented by quantitative verification.

Specification matching

With what probability does system behaviour match its specification?
Integrated analysis

**Qualitative** verification complemented by **quantitative** verification.

**Model checking**

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

Qualitative verification complemented by quantitative verification.

Model checking

For a given starting state how long is it until a given property $\phi$ holds?
Benefits of integration

■ Properties of the underlying mathematical structure may be deduced by the construction at the process algebra level.

■ Compositionality can be exploited both for model construction and (in some cases) for model analysis.

■ Formal reasoning techniques such as equivalence relations and model checking can be used to manipulate or interrogate models.

■ For example the congruence Markovian bisimulation, allows exact model reduction to be carried out compositionally.

■ Stochastic model checking based on the Continuous Stochastic Logic (CSL) allows automatic evaluation of quantified properties of the behaviour of the system.
Benefits of integration

- **Properties of the underlying mathematical structure** may be deduced by the construction at the process algebra level.

- **Compositionality** can be exploited both for model construction and (in some cases) for model analysis.

- Formal reasoning techniques such as equivalence relations and model checking can be used to manipulate or interrogate models.

- For example the congruence Markovian bisimulation, allows exact model reduction to be carried out compositionally.

- **Stochastic model checking** based on the Continuous Stochastic Logic (CSL) allows automatic evaluation of quantified properties of the behaviour of the system.
Benefits of integration

- **Properties of the underlying mathematical structure** may be deduced by the construction at the process algebra level.

- **Compositionality** can be exploited both for model construction and (in some cases) for model analysis.

- **Formal reasoning techniques** such as equivalence relations and model checking can be used to manipulate or interrogate models.

  - For example the congruence Markovian bisimulation, allows exact model reduction to be carried out compositionally.

  - **Stochastic model checking** based on the Continuous Stochastic Logic (CSL) allows automatic evaluation of quantified properties of the behaviour of the system.
Benefits of integration

- **Properties of the underlying mathematical structure** may be deduced by the construction at the process algebra level.

- **Compositionality** can be exploited both for model construction and (in some cases) for model analysis.

- **Formal reasoning techniques** such as equivalence relations and model checking can be used to manipulate or interrogate models.

- For example the congruence **Markovian bisimulation**, allows exact model reduction to be carried out **compositionally**.

- **Stochastic model checking** based on the Continuous Stochastic Logic (CSL) allows automatic evaluation of quantified properties of the behaviour of the system.
Benefits of integration

- Properties of the underlying mathematical structure may be deduced by the construction at the process algebra level.

- Compositionality can be exploited both for model construction and (in some cases) for model analysis.

- Formal reasoning techniques such as equivalence relations and model checking can be used to manipulate or interrogate models.

- For example the congruence Markovian bisimulation, allows exact model reduction to be carried out compositionally.

- Stochastic model checking based on the Continuous Stochastic Logic (CSL) allows automatic evaluation of quantified properties of the behaviour of the system.
Outline

1. Stochastic Process Algebras
2. Modelling in a Data Rich World
3. ProPPA
4. Inference
5. Results
6. Conclusions
There are many situations in which we wish to model and analyse behaviour of complex systems, which are operational and generate data, but which many not be completely transparent to us.

I will use the example of systems biology, where particular biological phenomena are observed and wet lab experiments can typically collect data on some parts of the system, but the basic mechanisms, or the parameters governing their behaviour, are unknown.
There are many situations in which we wish to model and analyse behaviour of complex systems, which are operational and generate data, but which may not be completely transparent to us.

I will use the example of systems biology, where particular biological phenomena are observed and wet lab experiments can typically collect data on some parts of the system, but the basic mechanisms, or the parameters governing their behaviour, are unknown.
### Molecular processes as concurrent computations

<table>
<thead>
<tr>
<th>Concurrency</th>
<th>Molecular Biology</th>
<th>Metabolism</th>
<th>Signal Transduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrent computational processes</td>
<td>Molecules</td>
<td>Enzymes and metabolites</td>
<td>Interacting proteins</td>
</tr>
<tr>
<td>Synchronous communication</td>
<td>Molecular interaction</td>
<td>Binding and catalysis</td>
<td>Binding and catalysis</td>
</tr>
<tr>
<td>Transition or mobility</td>
<td>Biochemical modification or relocation</td>
<td>Metabolite synthesis</td>
<td>Protein binding, modification or sequestration</td>
</tr>
</tbody>
</table>

Formal modelling in systems biology

- Formal languages provide a convenient interface for describing complex systems, reflecting what is known about the components and their behaviour.
- High-level abstraction eases writing and manipulating models.
- They are compiled into executable models which can be run to deepen understanding of the model.
- Formal nature lends itself to automatic, rigorous methods for analysis and verification.
- Executing the model generates data that can be compared with biological data.

...but what if parts of the system are unknown?

Bio-PEPA modelling

- The state of the system at any time consists of the local states of each of its “species” components, describing biochemical entities.

- The local states of components are quantitative rather than functional, i.e. biological changes to species are represented as distinct components.

- A component varying its state corresponds to it varying its amount through reactions modelled as interactions between components.

- The effect of a reaction is to vary the parameter of a component by a number corresponding to the stoichiometry of this species in the reaction.
Bio-PEPA modelling

- The state of the system at any time consists of the local states of each of its “species” components, describing biochemical entities.

- The local states of components are quantitative rather than functional, i.e. biological changes to species are represented as distinct components.

- A component varying its state corresponds to it varying its amount through reactions modelled as interactions between components.

- The effect of a reaction is to vary the parameter of a component by a number corresponding to the stoichiometry of this species in the reaction.
Bio-PEPA modelling

- The state of the system at any time consists of the local states of each of its “species” components, describing biochemical entities.

- The local states of components are quantitative rather than functional, i.e. biological changes to species are represented as distinct components.

- A component varying its state corresponds to it varying its amount through reactions modelled as interactions between components.

- The effect of a reaction is to vary the parameter of a component by a number corresponding to the stoichiometry of this species in the reaction.
Bio-PEPA modelling

- The state of the system at any time consists of the local states of each of its “species” components, describing biochemical entities.

- The local states of components are quantitative rather than functional, i.e. biological changes to species are represented as distinct components.

- A component varying its state corresponds to it varying its amount through reactions modelled as interactions between components.

- The effect of a reaction is to vary the parameter of a component by a number corresponding to the stoichiometry of this species in the reaction.
The semantics is defined by two transition relations:

- First, a capability relation — is a transition possible?

- Second, a stochastic relation — gives rate of a transition, derived from the parameters of the model.
Example

\[
\begin{align*}
   k_s &= 0.5; \\
   k_r &= 0.1; \\
   \text{kineticLawOf spread} &\colon k_s \times I \times S; \\
   \text{kineticLawOf stop1} &\colon k_r \times S \times S; \\
   \text{kineticLawOf stop2} &\colon k_r \times S \times R; \\
   I &= (\text{spread,1}) \downarrow; \\
   S &= (\text{spread,1}) \uparrow + (\text{stop1,1}) \downarrow + (\text{stop2,1}) \downarrow; \\
   R &= (\text{stop1,1}) \uparrow + (\text{stop2,1}) \uparrow; \\
   I[10] &\bowtie S[5] \bowtie R[0]
\end{align*}
\]
Example

\[
\begin{align*}
\text{k}_s &= 0.5; \\
\text{k}_r &= 0.1; \\
\text{kineticLawOf spread} &: \text{k}_s \ast \text{I} \ast \text{S}; \\
\text{kineticLawOf stop1} &: \text{k}_r \ast \text{S} \ast \text{S}; \\
\text{kineticLawOf stop2} &: \text{k}_r \ast \text{S} \ast \text{R}; \\
\text{I} &= (\text{spread},1) \downarrow; \\
\text{S} &= (\text{spread},1) \uparrow + (\text{stop1},1) \downarrow + (\text{stop2},1) \downarrow; \\
\text{R} &= (\text{stop1},1) \uparrow + (\text{stop2},1) \uparrow; \\
\text{I}[10] &\bowtie \text{S}[5] \bowtie \text{R}[0]
\end{align*}
\]
Usual process of parameterising a model is iterative and manual.
Alternative perspective

Model creation is data-driven
Machine Learning: Bayesian statistics

- Represent belief and uncertainty as probability distributions (prior, posterior).
- Treat parameters and unobserved variables similarly.
- Bayes’ Theorem:

\[ P(\theta \mid D) = \frac{P(\theta) \cdot P(D \mid \theta)}{P(D)} \]

posterior \(\propto\) prior \cdot likelihood
Thus there are two approaches to model construction:

**Machine Learning:** extracting a model from the data generated by the system, or refining a model based on system behaviour using statistical techniques.

**Mechanistic Modelling:** starting from a description or hypothesis, construct a formal model that algorithmically mimics the behaviour of the system, validated against data.
Comparing the techniques

Data-driven modelling:

+ rigorous handling of parameter uncertainty
  - limited or no treatment of stochasticity
  - in many cases bespoke solutions are required which can limit the size of system which can be handled

Mechanistic modelling:

+ general execution "engine" (deterministic or stochastic) can be reused for many models
+ models can be used speculatively to investigate roles of parameters, or alternative hypotheses
  - parameters are assumed to be known and fixed, or costly approaches must be used to seek appropriate parameterisation
Comparing the techniques

Data-driven modelling:

+ rigorous handling of parameter uncertainty
  - limited or no treatment of stochasticity
  - in many cases bespoke solutions are required which can limit the size of system which can be handled

Mechanistic modelling:

+ general execution "engine" (deterministic or stochastic) can be reused for many models
+ models can be used speculatively to investigate roles of parameters, or alternative hypotheses
  - parameters are assumed to be known and fixed, or costly approaches must be used to seek appropriate parameterisation
Developing a probabilistic programming approach

What if we could...

- include information about uncertainty in the model?
- automatically use observations to refine this uncertainty?
- do all this in a formal context?

Starting from the existing process algebra (Bio-PEPA), we have developed a new language ProPPA that addresses these issues.

Outline

1. Stochastic Process Algebras
2. Modelling in a Data Rich World
3. ProPPA
4. Inference
5. Results
6. Conclusions
Probabilistic programming

- A programming paradigm for describing incomplete knowledge scenarios, and resolving the uncertainty.

- Programs are probabilistic models in a high level language, like software code.

- Offers automated inference without the need to write bespoke solutions.

- Platforms: IBAL, Church, Infer.NET, Fun, Anglican, WebPPL,....

- Key actions: specify a distribution, specify observations, infer posterior distribution.
Probabilistic programming workflow

- **Describe how the data is generated** in syntax like a conventional programming language, but leaving some variables uncertain.

- **Specify observations**, which impose constraints on acceptable outputs of the program.

- **Run program forwards**: Generate data consistent with observations.

- **Run program backwards**: Find values for the uncertain variables which make the output match the observations.
Probabilistic programming workflow

- **Describe how the data is generated** in syntax like a conventional programming language, but leaving some variables uncertain.

- **Specify observations**, which impose constraints on acceptable outputs of the program.

- **Run program forwards**: Generate data consistent with observations.

- **Run program backwards**: Find values for the uncertain variables which make the output match the observations.
Probabilistic programming workflow

- **Describe how the data is generated** in syntax like a conventional programming language, but leaving some variables uncertain.

- **Specify observations**, which impose constraints on acceptable outputs of the program.

- **Run program forwards**: Generate data consistent with observations.

- **Run program backwards**: Find values for the uncertain variables which make the output match the observations.
Probabilistic programming workflow

- **Describe how the data is generated** in syntax like a conventional programming language, but leaving some variables uncertain.

- **Specify observations**, which impose constraints on acceptable outputs of the program.

- **Run program forwards**: Generate data consistent with observations.

- **Run program backwards**: Find values for the uncertain variables which make the output match the observations.
A Probabilistic Programming Process Algebra: ProPPA

The objective of ProPPA is to retain the features of the stochastic process algebra:

- simple model description in terms of components
- rigorous semantics giving an executable version of the model...

... whilst also incorporating features of a probabilistic programming language:

- recording uncertainty in the parameters
- ability to incorporate observations into models
- access to inference to update uncertainty based on observations
A Probabilistic Programming Process Algebra: ProPPA

The objective of ProPPA is to retain the features of the stochastic process algebra:

- simple model description in terms of components
- rigorous semantics giving an executable version of the model...

... whilst also incorporating features of a probabilistic programming language:

- recording uncertainty in the parameters
- ability to incorporate observations into models
- access to inference to update uncertainty based on observations
Example Revisited

$k_s = 0.5$;
$k_r = 0.1$;

kineticLawOf spread : $k_s \cdot I \cdot S$;
kineticLawOf stop1 : $k_r \cdot S \cdot S$;
kineticLawOf stop2 : $k_r \cdot S \cdot R$;

$I = (\text{spread},1) \downarrow$;
$S = (\text{spread},1) \uparrow + (\text{stop1},1) \downarrow + (\text{stop2},1) \downarrow$;
$R = (\text{stop1},1) \uparrow + (\text{stop2},1) \uparrow$;

$I[10] \bowtie S[5] \bowtie R[0]$
Additions

Declaring uncertain parameters:

- $k_s = \text{Uniform}(0,1)$;
- $k_t = \text{Uniform}(0,1)$;

Providing observations:

- `observe('trace')`

Specifying inference approach:

- `infer('ABC')`
k_s = Uniform(0,1);
k_r = Uniform(0,1);

kineticLawOf spread : k_s * I * S;
kineticLawOf stop1 : k_r * S * S;
kineticLawOf stop2 : k_r * S * R;

I = (spread,1) ↓ ;
S = (spread,1) ↑ + (stop1,1) ↓ + (stop2,1) ↓ ;
R = (stop1,1) ↑ + (stop2,1) ↑ ;


observe('trace')
infer('ABC') //Approximate Bayesian Computation
A Bio-PEPA model can be interpreted as a CTMC; however, CTMCs cannot capture uncertainty in the rates (every transition must have a concrete rate).

ProPPA models include uncertainty in the parameters, which translates into uncertainty in the transition rates.

A ProPPA model should be mapped to something like a distribution over CTMCs.
Stochastic Process Algebras
Modelling in a Data Rich World
ProPPA
Inference
Results
Conclusions

$k = 2$

model

CTMC
The parameter $k \in [0,5]$ defines the model, which is a set of CTMCs.
$k \sim p$

model

\[ \mu \]

distribution over CTMCs
Constraint Markov Chains (CMCs) are a generalization of DTMCs, in which the transition probabilities are not concrete, but can take any value satisfying some constraints.

A CMC is a tuple $\langle S, o, A, V, \phi \rangle$, where:

- $S$ is the set of states, of cardinality $k$.
- $o \in S$ is the initial state.
- $A$ is a set of atomic propositions.
- $V : S \rightarrow 2^A$ gives a set of acceptable labellings for each state.
- $\phi : S \times [0, 1]^k \rightarrow \{0, 1\}$ is the constraint function.

Caillard et al., Constraint Markov Chains, Theoretical Computer Science, 2011
In a CMC, arbitrary constraints are permitted, expressed through the function $\phi$: $\phi(s, \vec{p}) = 1$ iff $\vec{p}$ is an acceptable vector of transition probabilities from state $s$.

However,

- CMCs are defined only for the discrete-time case, and
- this does not say anything about how likely a value is to be chosen, only about whether it is acceptable.

To address these shortcomings, we define **Probabilistic Constraint Markov Chains**.
A Probabilistic Constraint Markov Chain is a tuple \( \langle S, o, A, V, \phi \rangle \), where:

- \( S \) is the set of states, of cardinality \( k \).
- \( o \in S \) is the initial state.
- \( A \) is a set of atomic propositions.
- \( V : S \rightarrow 2^A \) gives a set of acceptable labellings for each state.
- \( \phi : S \times [0, \infty)^k \rightarrow [0, \infty) \) is the constraint function.

- This is applicable to continuous-time systems.
- \( \phi(s, \cdot) \) is now a probability density function on the transition rates from state \( s \).
Semantics of ProPPA

The semantics definition follows that of Bio-PEPA, which is defined using two transition relations:

- Capability relation — is a transition possible?
- Stochastic relation — gives distribution of the rate of a transition

The distribution over the parameter values induces a distribution over transition rates.

Rules are expressed as state-to-function transition systems (FuTS\(^1\)).

This gives rise the underlying PCMC.

\(^1\)De Nicola et al., *A Uniform Definition of Stochastic Process Calculi*, ACM Computing Surveys, 2013
Semantics of ProPPA

The semantics definition follows that of Bio-PEPA, which is defined using two transition relations:

- Capability relation — is a transition possible?
- Stochastic relation — gives distribution of the rate of a transition

The distribution over the parameter values induces a distribution over transition rates.

Rules are expressed as state-to-function transition systems (FuTS\(^1\)).

This gives rise the underlying PCMC.

\(^1\)De Nicola et al., A Uniform Definition of Stochastic Process Calculi, ACM Computing Surveys, 2013
Simulating Probabilistic Constraint Markov Chains

Probabilistic Constraint Markov Chains are open to two alternative dynamic interpretations:

1. **Uncertain Markov Chains**: For each trajectory, for each uncertain transition rate, sample once at the start of the run and use that value throughout;

2. **Imprecise Markov Chains**: During each trajectory, each time a transition with an uncertain rate is encountered, sample a value but then discard it and re-sample whenever this transition is visited again.

Our current work is focused on the Uncertain Markov Chain case.
Probabilistic Constraint Markov Chains are open to two alternative dynamic interpretations:

1. **Uncertain Markov Chains**: For each trajectory, for each uncertain transition rate, sample once at the start of the run and use that value throughout;

2. **Imprecise Markov Chains**: During each trajectory, each time a transition with an uncertain rate is encountered, sample a value but then discard it and re-sample whenever this transition is visited again.

Our current work is focused on the Uncertain Markov Chain case.
Inference

\[ k \sim p \]

model

\[ \mu \]

distribution over CTMCs
Inference

\[ k \sim p \]

model

\[ \mu \]

distribution over CTMCs

inference

observations

\[ \mu^* \]

posterior distribution
Inference

\[ P(\theta \mid D) \propto P(\theta)P(D \mid \theta) \]

- Exact inference is impossible, as we cannot calculate the likelihood.
- We must use approximate algorithms or approximations of the system.
- The ProPPA semantics does not define a single inference algorithm, allowing for a modular approach.
- Different algorithms can act on different input (time-series vs properties), return different results or in different forms.
Inferring likelihood in uncertain CTMCs

Transient state probabilities can be expressed as:

\[
\frac{dp_i(t)}{dt} = \sum_{j \neq i} p_j(t) \cdot q_{ji} - p_i(t) \sum_{j \neq i} q_{ij}
\]

The probability of a single observation \((y, t)\) can then be expressed as

\[
p(y, t) = \sum_{i \in S} p_i(t) \pi(y \mid i)
\]

where \(\pi(y \mid i)\) is the probability of observing \(y\) when in state \(i\).

The likelihood can then be expressed as

\[
P(D \mid \theta) = \prod_{j=1}^{N} \sum_{i \in S} p_{i \mid \theta}(t_j) \pi(y_j \mid i)
\]
Inferring likelihood in uncertain CTMCs

Transient state probabilities can be expressed as:

\[
\frac{dp_i(t)}{dt} = \sum_{j \neq i} p_j(t) \cdot q_{ji} - p_i(t) \sum_{j \neq i} q_{ij}
\]

The probability of a single observation \((y, t)\) can then be expressed as

\[
p(y, t) = \sum_{i \in S} p_i(t) \pi(y \mid i)
\]

where \(\pi(y \mid i)\) is the probability of observing \(y\) when in state \(i\).

The likelihood can then be expressed as

\[
P(D \mid \theta) = \prod_{j=1}^{N} \sum_{i \in S} p_{(i|\theta)}(t_j) \pi(y_j \mid i)
\]
Inferring likelihood in uncertain CTMCs

Transient state probabilities can be expressed as:

\[
\frac{dp_i(t)}{dt} = \sum_{j \neq i} p_j(t) \cdot q_{ji} - p_i(t) \sum_{j \neq i} q_{ij}
\]

The probability of a single observation \((y, t)\) can then be expressed as

\[
p(y, t) = \sum_{i \in S} p_i(t) \pi(y \mid i)
\]

where \(\pi(y \mid i)\) is the probability of observing \(y\) when in state \(i\).

The likelihood can then be expressed as

\[
P(D \mid \theta) = \prod_{j=1}^{N} \sum_{i \in S} p_{(i \mid \theta)}(t_j) \pi(y_j \mid i)
\]
Calculating the transient probabilities

For finite state-spaces, the transient probabilities can, in principle, be computed as

\[ p(t) = p(0)e^{Qt}. \]

Likelihood is hard to compute:

- Computing \( e^{Qt} \) is expensive if the state space is large
- Impossible directly in infinite state-spaces
Basic Inference

- Approximate Bayesian Computation is a simple simulation-based solution:
  - Approximates posterior distribution over parameters as a set of samples
  - Likelihood of parameters is approximated with a notion of distance.
Approximate Bayesian Computation is a simple simulation-based solution:

- Approximates posterior distribution over parameters as a set of samples
- Likelihood of parameters is approximated with a notion of distance.
Approximate Bayesian Computation is a simple simulation-based solution:
- Approximates posterior distribution over parameters as a set of samples
- Likelihood of parameters is approximated with a notion of distance.
Basic Inference

- Approximate Bayesian Computation is a simple simulation-based solution:
  - Approximates posterior distribution over parameters as a set of samples
  - Likelihood of parameters is approximated with a notion of distance.
Basic Inference

- Approximate Bayesian Computation is a simple simulation-based solution:
  - Approximates posterior distribution over parameters as a set of samples
  - Likelihood of parameters is approximated with a notion of distance.

\[
\sum (x_i - y_i)^2 > \varepsilon \quad \text{rejected}
\]
Basic Inference

- Approximate Bayesian Computation is a simple simulation-based solution:
  - Approximates posterior distribution over parameters as a set of samples
  - Likelihood of parameters is approximated with a notion of distance.

\[ \sum (x_i - y_i)^2 < \varepsilon \]
accepted
Approximate Bayesian Computation

ABC algorithm

1. Sample a parameter set from the prior distribution.
2. Simulate the system using these parameters.
3. Compare the simulation trace obtained with the observations.
4. If distance $< \epsilon$, accept, otherwise reject.

This results in an approximation to the posterior distribution. As $\epsilon \to 0$, set of samples converges to true posterior. We use a more elaborate version based on Markov Chain Monte Carlo sampling.
Inference for infinite state spaces

Various methods become inefficient or inapplicable as the state-space grows.

How to deal with unbounded systems?

- Multiple simulation runs
- Large population approximations (diffusion, Linear Noise, . . .)
- Systematic truncation
- **Random truncations**
Expanding the likelihood

The likelihood can be written as an infinite series:

\[ p(x', t' \mid x, t) = \sum_{N=0}^{\infty} p^{(N)}(x', t' \mid x, t) \]

\[ = \sum_{N=0}^{\infty} \left[ f^{(N)}(x', t' \mid x, t) - f^{(N-1)}(x', t' \mid x, t) \right] \]

where

- \( x^* = \max\{x, x'\} \)
- \( p^{(N)}(x', t' \mid x, t) \) is the probability of going from state \( x \) at time \( t \) to state \( x' \) at time \( t' \) through a path with maximum state \( x^* + N \)
- \( f^{(N)} \) is the same, except the maximum state cannot exceed \( x^* + N \) (but does not have to reach it)

Using Russian Roulette truncation we can estimate the infinite sum with a random truncation.
Expanding the likelihood

The likelihood can be written as an infinite series:

\[ p(x', t' \mid x, t) = \sum_{N=0}^{\infty} p^{(N)}(x', t' \mid x, t) = \sum_{N=0}^{\infty} \left[ f^{(N)}(x', t' \mid x, t) - f^{(N-1)}(x', t' \mid x, t) \right] \]

where

- \( x^* = \max\{x, x'\} \)
- \( p^{(N)}(x', t' \mid x, t) \) is the probability of going from state \( x \) at time \( t \) to state \( x' \) at time \( t' \) through a path with maximum state \( x^* + N \)
- \( f^{(N)} \) is the same, except the maximum state cannot exceed \( x^* + N \) (but does not have to reach it)

Using Russian Roulette truncation we can estimate the infinite sum with a random truncation.
Stochastic Process Algebras Modelling in a Data Rich World ProPPA Inference Results Conclusions

(3,0) (3,1) (3,2) (3,3) (3,4)
(2,0) (2,1) (2,2) (2,3) (2,4)
(1,0) (1,1) x (1,2) (1,3) (1,4)
(0,0) (0,1) (0,2) x' (0,3) (0,4)
Example model

\begin{align*}
k_s &= \text{Uniform}(0,1); \\
k_r &= \text{Uniform}(0,1); \\
\text{kineticLawOf spread} & : k_s \ast I \ast S; \\
\text{kineticLawOf stop1} & : k_r \ast S \ast S; \\
\text{kineticLawOf stop2} & : k_r \ast S \ast R; \\
I &= (\text{spread,1}) \downarrow; \\
S &= (\text{spread,1}) \uparrow + (\text{stop1,1}) \downarrow + (\text{stop2,1}) \downarrow; \\
R &= (\text{stop1,1}) \uparrow + (\text{stop2,1}) \uparrow; \\
I[10] & \bowtie S[5] \bowtie R[0] \\
\text{observe(’trace’)} \\
\text{infer(’ABC’) //Approximate Bayesian Computation}
\end{align*}
Results

Tested on the rumour-spreading example, giving the two parameters uniform priors.

- Approximate Bayesian Computation
- Returns posterior as a set of points (samples)
- Observations: time-series (single simulation)
Results: ABC
Results: ABC
Results: ABC

![Histogram of kr and ks values](image)

- Number of samples for kr and ks values.

- The graphs show the distribution of kr and ks values with the number of samples on the y-axis and the values on the x-axis.
Genetic Toggle Switch

- Two mutually-repressing genes: promoters (unobserved) and their protein products
- Bistable behaviour: switching induced by environmental changes
- Synthesised in *E. coli*\(^2\)
- Stochastic variant\(^3\) where switching is induced by noise

---


\(^3\) Tian & Burrage, *Stochastic models for regulatory networks of the genetic toggle switch*, PNAS, 2006
Genetic Toggle Switch

\[ \text{\( P_1 \)} \quad \text{\( P_2 \)} \]

\[ \text{\( G_{1,\text{on}} \)} \quad \text{\( G_{1,\text{off}} \)} \quad \text{\( G_{2,\text{on}} \)} \quad \text{\( G_{2,\text{off}} \)} \]

\[ \text{\( \varnothing \)} \quad \text{\( \varnothing \)} \quad \text{\( \varnothing \)} \quad \text{\( \varnothing \)} \]

\[ \text{\( \text{participates} \)} \quad \text{\( \text{accelerates} \)} \]
Toggle switch model: species

\begin{align*}
  G1 &= \text{activ1} \uparrow + \text{deact1} \downarrow + \text{expr1} \oplus; \\
  G2 &= \text{activ2} \uparrow + \text{deact2} \downarrow + \text{expr2} \oplus; \\
  P1 &= \text{expr1} \uparrow + \text{degr1} \downarrow + \text{deact2} \oplus; \\
  P2 &= \text{expr2} \uparrow + \text{degr2} \downarrow + \text{deact1} \oplus \\
\end{align*}

\text{observe}(\text{toggle\_obs}); \\
\text{infer}(\text{rouletteGibbs});
\[ \theta_1 = \text{Gamma}(3,5); \ //\text{etc...} \]

\[
\text{kineticLawOf } \text{expr1} : \theta_1 \times G_1; \\
\text{kineticLawOf } \text{expr2} : \theta_2 \times G_2; \\
\text{kineticLawOf } \text{degr1} : \theta_3 \times P_1; \\
\text{kineticLawOf } \text{degr2} : \theta_4 \times P_2; \\
\text{kineticLawOf } \text{activ1} : \theta_5 \times (1 - G_1); \\
\text{kineticLawOf } \text{activ2} : \theta_6 \times (1 - G_2); \\
\text{kineticLawOf } \text{deact1} : \theta_7 \times \exp(r \ast P_2) \times G_1; \\
\text{kineticLawOf } \text{deact2} : \theta_8 \times \exp(r \ast P_1) \times G_2; \\
\]

\[
G_1 = \text{activ1} \uparrow + \text{deact1} \downarrow + \text{expr1} \oplus; \\
G_2 = \text{activ2} \uparrow + \text{deact2} \downarrow + \text{expr2} \oplus; \\
P_1 = \text{expr1} \uparrow + \text{degr1} \downarrow + \text{deact2} \oplus; \\
P_2 = \text{expr2} \uparrow + \text{degr2} \downarrow + \text{deact1} \oplus \\
\]

\[
G_1[1] \bowtie G_2[0] \bowtie P_1[20] \bowtie P_2[0] \\
\]

\[
\text{observe(toggle_{obs});} \\
\text{infer(rouletteGibbs);} \\
\]
Experiment

- Simulated observations
- Gamma priors on all parameters (required by algorithm)
- Goal: learn posterior of 8 parameters
- 5000 samples taken using the Gibbs-like random truncation algorithm
Genes (unobserved)
Proteins
Observations used
Results
Outline

1. Stochastic Process Algebras
2. Modelling in a Data Rich World
3. ProPPA
4. Inference
5. Results
6. Conclusions
Workflow

- **model**
- compile
- low-level description
- infer
- inference algorithm
- inference results (samples)
  - statistics
  - plotting
  - prediction
  - ...

Stochastic Process Algebras Modelling in a Data Rich World

ProPPA

Inference

Results

Conclusions
Summary

- ProPPA is a process algebra that incorporates uncertainty and observations directly in the model, influenced by probabilistic programming.
- Syntax remains similar to Bio-PEPA.
- Semantics defined in terms of an extension of Constraint Markov Chains.
- Observations can be either time-series or logical properties.
- Parameter inference based on random truncations (Russian Roulette) offers new possibilities for inference.
Challenges and Future Directions

- The value of observations

Can we reason about the “distance” between $\mu$ and $\mu^*$?
Challenges and Future Directions

- Heterogeneous populations

What if we are seeking the “optimal mix” rather than the best individual representative?
Thanks

Anastasios Georgoulas

Guido Sanguinetti

Funding from ERC, Microsoft Research and CEC FET-Proactive Programme FoCAS.
Thanks

- Anastasis Georgoulas
- Guido Sanguinetti

Funding from ERC, Microsoft Research and CEC FET-Proactive Programme FoCAS.