SPAs for performance modelling: Lecture 8 — Stochastic Probes

Jane Hillston (+ Allan Clark)

LFCS, School of Informatics
The University of Edinburgh
Scotland

17th April 2013

THE UNIVERSITY
of EDINBURGH

Outline

1. Stochastic probes
2. Passage Time and Passage End Analysis
3. Models with a Spatial Aspect
   - Spatial Challenge: Capturing physical space

Outline

1 Stochastic probes

2 Passage Time and Passage End Analysis

3 Models with a Spatial Aspect
- Spatial Challenge: Capturing physical space

Introduction: Stochastic probes

Last week we saw that it was possible to query models using CSL formulae and the stochastic model checker PRISM.

Introduction: Stochastic probes

Last week we saw that it was possible to query models using CSL formulae and the stochastic model checker PRISM.

This is a very powerful technique but it takes quite a lot of skill to write CSL formulae that capture just what you want to ask.

Introduction: Stochastic probes

Last week we saw that it was possible to query models using CSL formulae and the stochastic model checker PRISM.

This is a very powerful technique but it takes quite a lot of skill to write CSL formulae that capture just what you want to ask.

In this lecture we will look at the alternative approach of using stochastic probes (and extended stochastic probes XSP).

Performance Measure Probes

- A performance measurement probe is a separate PEPA component which may be attached to the model to be measured.

Performance Measure Probes

- A performance measurement probe is a separate PEPA component which may be attached to the model to be measured.
- The probe component observes the actions of the measured model and changes its state accordingly

Performance Measure Probes

- A performance measurement probe is a separate PEPA component which may be attached to the model to be measured.
- The probe component observes the actions of the measured model and changes its state accordingly
- To measure the model the probe can then be interrogated for its state.

Performance Measure Probes

- A performance measurement probe is a separate PEPA component which may be attached to the model to be measured.
- The probe component observes the actions of the measured model and changes its state accordingly
- To measure the model the probe can then be interrogated for its state.
- Sometimes the probes are created implicitly by the tool in order to carry out a calculation.

Simple passage model: making tea

```
Idle    = (begin        , beginRate) . Water  ;
Water   = (fillKettle   , fillRate)  . TurnOn ;
TurnOn  = (turnOnKettle, onRate)     . Wait   ;
Wait    = (boil         , boilRate)  . Pour   ;
Pour    = (pour         , pourRate)  . Milk   ;
AddMilk = (milk         , milkRate)  . Stir   ;
Stir    = (stir         , stirRate)  . Drink  ;
Drink   = (drink        , drinkRate) . Idle   ;

Idle
```

A Simple Measurement

- Suppose we wish to find out what the probability that the person in the model is currently making a cup of tea.

A Simple Measurement

- Suppose we wish to find out what the probability that the person in the model is currently making a cup of tea.
- One possibility is to simply add up the probabilities of the states in which the person is making their tea.

A Simple Measurement

- Suppose we wish to find out what the probability that the person in the model is currently making a cup of tea.
- One possibility is to simply add up the probabilities of the states in which the person is making their tea.
- However this approach is not very robust because if the model is revised this measurement must also be revised.

Simple passage probe

```
ProbeIdle    = (begin, ⊤) . ProbeRunning ;

ProbeRunning = (stir , ⊤) . ProbeIdle ;
```

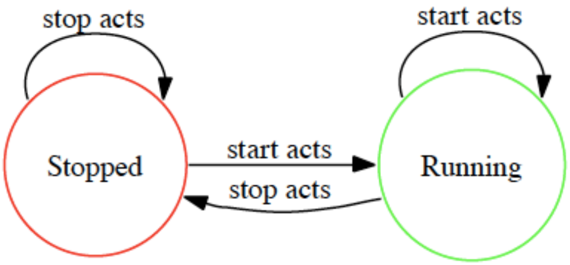
$\texttt{Idle} \underset{\{begin, stir\}}{\bowtie} \texttt{ProbeIdle}$

Simple passage probe

```
ProbeIdle    = (begin, ⊤) . ProbeRunning
             + (stir , ⊤) . ProbeIdle     ;
ProbeRunning = (stir , ⊤) . ProbeIdle
             + (begin, ⊤) . ProbeRunning ;
```

$\texttt{Idle} \underset{\{begin, stir\}}{\bowtie} \texttt{ProbeIdle}$

Generic probe graph



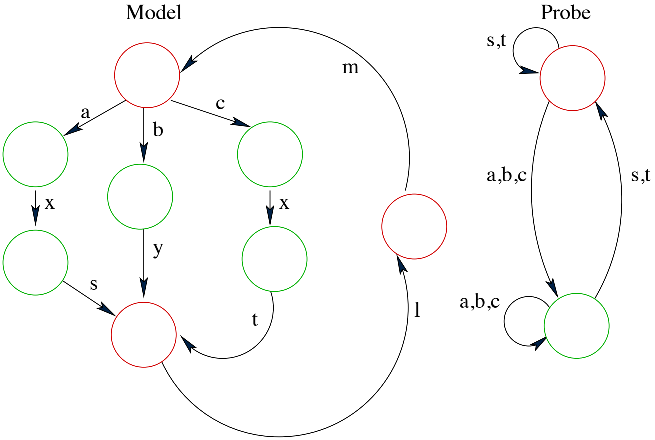
A Simple Measurement

- Now we need only measure the probability that the probe is in the state *ProbeRunning*.

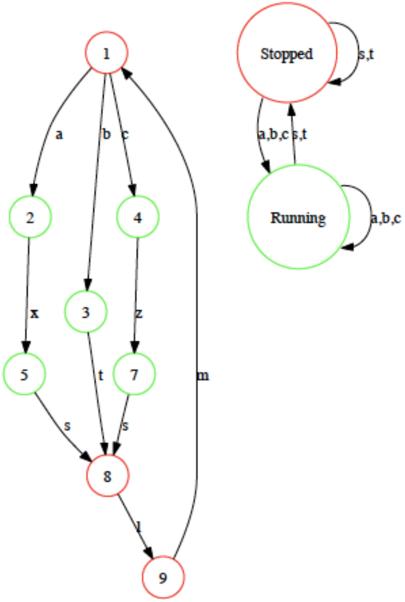
A Simple Measurement

- Now we need only measure the probability that the probe is in the state *ProbeRunning*.
- If the model is revised the probe need not necessarily be revised (unless it directly affects the measurement in question).

A probe graph



A probe graph



The boiler definitions

```
Boiler    = (cool  , coolRate  ) . Boiling
          + (pour  , ⊤         ) . Refilling
          ;
Boiling   = (boil  , boilRate  ) . Boiler  ;
Refilling = (refill, refillRate) . Boiling ;
```

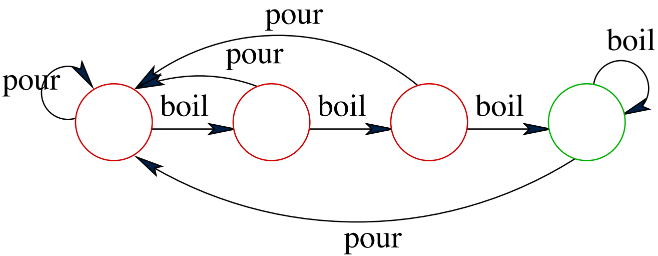
```
Idle    = (begin       , teaRate)   . Water  ;
Pour    = (pour        , pourRate)  . Milk   ;
AddMilk = (milk        , milkRate)  . Stir   ;
Stir    = (stir        , stirRate)  . Drink  ;
Drink   = (drink       , drinkRate) . Idle   ;
```

$\texttt{Idle}[30] \underset{\{pour\}}{\bowtie} \texttt{Boiler}$

A more complex probe

Suppose we wish to ask the probability of being in a state which has seen the boiler re-boil three or more times without a tea drinker taking some water.

The corresponding probe graph



The boiler probe definitions

```
Probe0 = (boil, ⊤) . Probe1 ;

Probe1 = (boil, ⊤) . Probe2 ;

Probe2 = (boil, ⊤) . Probe3 ;


Probe3 = (pour, ⊤) . Probe0 ;
```

$$\texttt{Probe0} \underset{\{boil,pour\}}{\bowtie} (\texttt{Idle} \underset{\{pour\}}{\bowtie} \texttt{Boiler})$$

The boiler probe definitions

$$\mathtt{Probe0} = (\mathtt{boil}, \top) . \mathtt{Probe1} ;$$

$$\begin{aligned}\mathtt{Probe1} &= (\mathtt{boil}, \top) . \mathtt{Probe2} ; \\ &+ (\mathtt{pour}, \top) . \mathtt{Probe0} ; \\ \mathtt{Probe2} &= (\mathtt{boil}, \top) . \mathtt{Probe3} ; \\ &+ (\mathtt{pour}, \top) . \mathtt{Probe0} ;\end{aligned}$$

$$\mathtt{Probe3} = (\mathtt{pour}, \top) . \mathtt{Probe0} ;$$

$$\mathtt{Probe0} \underset{\{boil, pour\}}{\bowtie} (\mathtt{Idle} \underset{\{pour\}}{\bowtie} \mathtt{Boiler})$$

The boiler probe definitions

$$
\begin{aligned}
\texttt{Probe0} &= (\texttt{boil}, \top) . \texttt{Probe1} ; \\
&\quad + (\texttt{pour}, \top) . \texttt{Probe0} ; \\
\texttt{Probe1} &= (\texttt{boil}, \top) . \texttt{Probe2} ; \\
&\quad + (\texttt{pour}, \top) . \texttt{Probe0} ; \\
\texttt{Probe2} &= (\texttt{boil}, \top) . \texttt{Probe3} ; \\
&\quad + (\texttt{pour}, \top) . \texttt{Probe0} ;
\end{aligned}
$$

$$
\begin{aligned}
\texttt{Probe3} &= (\texttt{pour}, \top) . \texttt{Probe0} \\
&\quad + (\texttt{boil}, \top) . \texttt{Probe3} ;
\end{aligned}
$$

$$
\texttt{Probe0} \underset{\{boil, pour\}}{\bowtie} (\texttt{Idle} \underset{\{pour\}}{\bowtie} \texttt{Boiler})
$$

Writing probes as regular expressions

Writing a set of probe definitions as PEPA definitions is error prone. In particular it is hard to get the self-loops correct.

- Instead we allow a regular expression-like language for defining probes in the ipc tool (Imperial PEPA compiler).

Writing probes as regular expressions

Writing a set of probe definitions as PEPA definitions is error prone. In particular it is hard to get the self-loops correct.

- Instead we allow a regular expression-like language for defining probes in the ipc tool (Imperial PEPA compiler).
- The previous probe would be written as:
  $((boil, boil, boil)/pour) : start, pour : stop)$

Writing probes as regular expressions

Writing a set of probe definitions as PEPA definitions is error prone. In particular it is hard to get the self-loops correct.

- Instead we allow a regular expression-like language for defining probes in the ipc tool (Imperial PEPA compiler).
- The previous probe would be written as:
  $((boil, boil, boil)/pour) : start, pour : stop)$
- The 'start' and 'stop' labels indicate that we are entering or exiting the state of the probe/model in which we are interested.

Writing probes as regular expressions

Writing a set of probe definitions as PEPA definitions is error prone. In particular it is hard to get the self-loops correct.

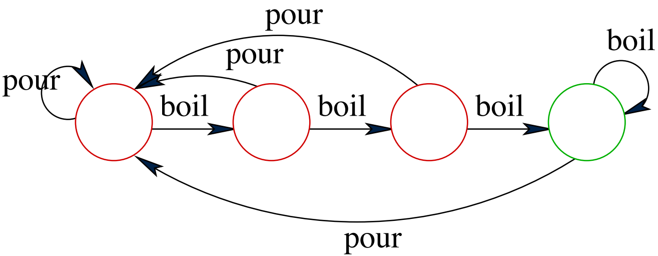
- Instead we allow a regular expression-like language for defining probes in the ipc tool (Imperial PEPA compiler).
- The previous probe would be written as: $((boil, boil, boil)/pour) : start, pour : stop)$
- The 'start' and 'stop' labels indicate that we are entering or exiting the state of the probe/model in which we are interested.
- This is then automatically translated into the PEPA component that is the probe and attached to the model.

Writing probes as regular expressions

Writing a set of probe definitions as PEPA definitions is error prone. In particular it is hard to get the self-loops correct.

- Instead we allow a regular expression-like language for defining probes in the ipc tool (Imperial PEPA compiler).
- The previous probe would be written as:
  $((\mathit{boil}, \mathit{boil}, \mathit{boil})/\mathit{pour}) : \mathit{start}, \mathit{pour} : \mathit{stop})$
- The 'start' and 'stop' labels indicate that we are entering or exiting the state of the probe/model in which we are interested.
- This is then automatically translated into the PEPA component that is the probe and attached to the model.
- In addition this allows us to separate probes from the model and attach several probes to a single model, either separately or simultaneously.

The corresponding probe graph



A regular-expression-like syntax for probes”

R	$:=$	*activity*	Observe action
	$\mid$	R_1, R_2	sequence
	$\mid$	$R_1 \mid R_2$	choice
	$\mid$	$R : label$	labelled
	$\mid$	$R\ n$	iterate n times
	$\mid$	$R\{m, n\}$	iterate between m and n times
	$\mid$	$R+$	one or more
	$\mid$	$R*$	zero or more
	$\mid$	$R?$	zero or one
	$\mid$	$R/activity$	observe R without the activity

Probe Implementation

1. Translate probe expression into Nondeterministic Finite Automaton (NFA)

Probe Implementation

1. Translate probe expression into Nondeterministic Finite Automaton (NFA)
2. Construct the Deterministic Finite Automaton (DFA) corresponding to the NFA

Probe Implementation

1. Translate probe expression into Nondeterministic Finite Automaton (NFA)
2. Construct the Deterministic Finite Automaton (DFA) corresponding to the NFA
3. Minimise the DFA

Probe Implementation

1. Translate probe expression into Nondeterministic Finite Automaton (NFA)
2. Construct the Deterministic Finite Automaton (DFA) corresponding to the NFA
3. Minimise the DFA
4. Add necessary self-loops to the DFA

Probe Implementation

1. Translate probe expression into Nondeterministic Finite Automaton (NFA)
2. Construct the Deterministic Finite Automaton (DFA) corresponding to the NFA
3. Minimise the DFA
4. Add necessary self-loops to the DFA
5. Pretty print as a PEPA component

Constructing probes

- Now, rather than a probe constructed as an additional PEPA component by hand, we think of the probe specification in terms of a regular expression of action types.

Constructing probes

- Now, rather than a probe constructed as an additional PEPA component by hand, we think of the probe specification in terms of a regular expression of action types.
- Consider the following probe:
  $a : start, (b, c, d), e : stop$

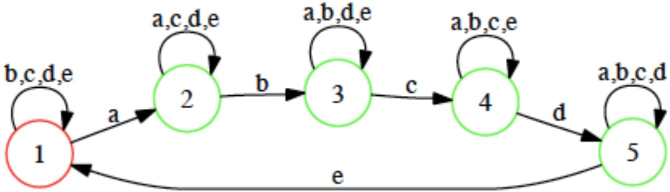
Constructing probes

- Now, rather than a probe constructed as an additional PEPA component by hand, we think of the probe specification in terms of a regular expression of action types.
- Consider the following probe:
  $a : start, (b, c, d), e : stop$
- This asks the question, what is the expected time from first observing an a activity, to observing a (possibly interrupted) sequence of activities b, c, d, e?

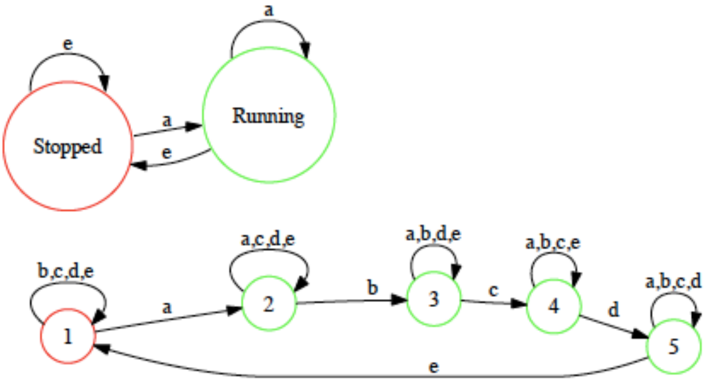
Constructing probes

- Now, rather than a probe constructed as an additional PEPA component by hand, we think of the probe specification in terms of a regular expression of action types.
- Consider the following probe:
  $a : start, (b, c, d), e : stop$
- This asks the question, what is the expected time from first observing an a activity, to observing a (possibly interrupted) sequence of activities b, c, d, e?
- The 'middle' part b, c, d may be some arbitrarily complex probe.

The probe graph



A probe on the probe



We can attach a master probe to an inner probe to retain the property of having only one running state.

Non-unique start and stop actions

- What if the probe may perform any of the start actions without wishing to move to the running state?
  Consider the following probe:
  $a, a, a : start, b : stop$

Non-unique start and stop actions

- What if the probe may perform any of the start actions without wishing to move to the running state?
  Consider the following probe:
  $a, a, a : start, b : stop$

- Or equivalently any of the stop actions without moving to the stopped state. Consider the following probe:
  $a : start, b, b, b : stop$

Non-unique start and stop actions

- What if the probe may perform any of the start actions without wishing to move to the running state?
  Consider the following probe:
  $a, a, a : start, b : stop$
- Or equivalently any of the stop actions without moving to the stopped state. Consider the following probe:
  $a : start, b, b, b : stop$
- Or both:
  $a : start, (a, a), a : stop$

Non-unique start and stop actions

- What if the probe may perform any of the start actions without wishing to move to the running state?
  Consider the following probe:
  $a, a, a : start, b : stop$

- Or equivalently any of the stop actions without moving to the stopped state. Consider the following probe:
  $a : start, b, b, b : stop$

- Or both:
  $a : start, (a, a), a : stop$

- This asks the very simple question: How long can we expect the model to take to perform four *a* actions?

Introducting immediate actions

The solution to this problem is to introduce immediate actions not to the general language of PEPA but only within the definition of probes.

Introducting immediate actions

The solution to this problem is to introduce immediate actions not to the general language of PEPA but only within the definition of probes.

- A *start* or *stop* label can then be implemented as an immediate action.

Introducting immediate actions

The solution to this problem is to introduce immediate actions not to the general language of PEPA but only within the definition of probes.

- A *start* or *stop* label can then be implemented as an immediate action.
- The probe $a : start$ becomes the prefix component: $(a, \top).(start, 1 : immediate).R$ where R is the probe's running state.

Introducting immediate actions

The solution to this problem is to introduce immediate actions not to the general language of PEPA but only within the definition of probes.

- A *start* or *stop* label can then be implemented as an immediate action.
- The probe $a : start$ becomes the prefix component: $(a, \top).(start, 1 : immediate).R$ where R is the probe's running state.
- For the remainder we'll abbreviate $(start, 1 : immediate).R$ as $start.R$

Standard master probe

```
ProbeStopped  = start . ProbeRunning
              + stop . ProbeStopped ;

ProbeRunning  = stop . ProbeStopped
              + start . ProbeRunning ;
```

Global probes vs. Local probes

So far we have only considered global probes, i.e. probes which are attached, externally to the PEPA model, observing all the behaviour.

Global probes vs. Local probes

So far we have only considered global probes, i.e. probes which are attached, externally to the PEPA model, observing all the behaviour.

Sometimes there are problems with global probes because they "see too much".

Global probes vs. Local probes

So far we have only considered global probes, i.e. probes which are attached, externally to the PEPA model, observing all the behaviour.

Sometimes there are problems with global probes because they "see too much".

This is illustrated by the following example based on a Client-Server system

Client-Server System

```
Client     = (work     , workRate    ) . ClientReq  ;
ClientReq  = (request  , requestRate) . ClientWait ;
ClientWait = (response, ⊤           ) . Client     ;

ServerIdle = (request  , ⊤             ) . ServerComp ;
ServerComp = (compute  , computeServer ) . ServerResp ;
ServerResp = (response, responseServer) . ServerIdle ;
```

$\texttt{Client}[4] \underset{\{\mathit{request}, \mathit{response}\}}{\bowtie} \texttt{ServerIdle}[2]$

Probing the Client-Server System

We may wish to ask: What is the expected response time?

This is a passage-time query which we might expect to be answered with the following probe specification.

$request : start, response : stop$

Client-Server System with Probe

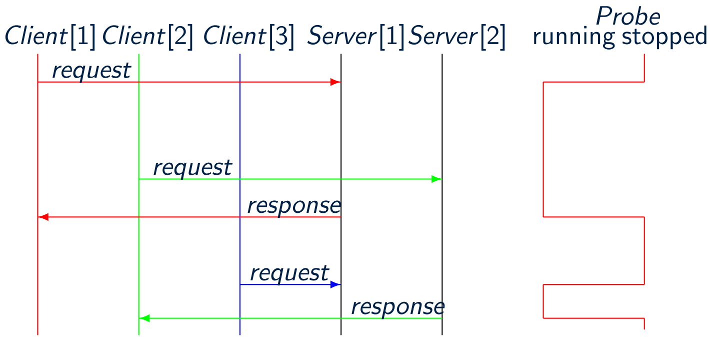
```
Client     = (work     , workRate     ) . ClientReq  ;
ClientReq  = (request  , requestRate) . ClientWait ;
ClientWait = (response, ⊤           ) . Client     ;

ServerIdle = (request  , ⊤              ) . ServerComp ;
ServerComp = (compute  , computeServer ) . ServerResp ;
ServerResp = (response, responseServer) . ServerIdle ;

Probe0     = (request, ⊤) . Probe1
           + (response, ⊤) . Probe0 ;
Probe1     = (response, ⊤). Probe0
           + (request, ⊤). Probe1 ;
```

$$(\texttt{Client}[4] \underset{\{request,\,response\}}{\bowtie} \texttt{ServerIdle}[2]) \underset{\{request,\,response\}}{\bowtie} \texttt{Probe0}$$

The problem with global probes



Using local probes

This problem can be fixed by placing the probe local to just one of the clients:

Using local probes

This problem can be fixed by placing the probe local to just one of the clients:

$$(\, (\, \texttt{Client} \underset{\{request, response\}}{\bowtie} \texttt{Probe0} \,) \bowtie \texttt{Client[3]} \,) \underset{\{request, response\}}{\bowtie} \texttt{ServerIdle[2]}$$

Using local probes

This problem can be fixed by placing the probe local to just one of the clients:

$$\left(\, \left(\, \texttt{Client} \underset{\{request,response\}}{\bowtie} \texttt{Probe0} \,\right) \bowtie \texttt{Client}[3] \,\right) \underset{\{request,response\}}{\bowtie} \texttt{ServerIdle}[2]$$

The probe specification language for ipc allows the user to place a probe using the :: notation, the model shown was generated with the following probe.

$$Client :: (request : start, response : stop)$$

Generalising the start and stop labels

Probes previously relied on the start and stop labels as being special such that the compiler could recognise them.

Generalising the start and stop labels

Probes previously relied on the start and stop labels as being special such that the compiler could recognise them.

Since probes may now 'send' *start* and *stop* communication signals using immediate actions we can generalise these labels to enable the sending of arbitrary signals.

Generalising the start and stop labels

Probes previously relied on the start and stop labels as being special such that the compiler could recognise them.

Since probes may now 'send' *start* and *stop* communication signals using immediate actions we can generalise these labels to enable the sending of arbitrary signals.

In addition since we have the ability to localise a probe we may use such communication labels to communicate important events from a local probe to a master probe.

Local and master probes

We may now have several local probes which communicate with a control probe.

$$(((P \underset{\mathcal{N}}{\bowtie} ProbeP) \underset{\mathcal{K}}{\bowtie} Q) \underset{\mathcal{L}}{\bowtie} (R \underset{\mathcal{M}}{\bowtie} (S \underset{\mathcal{O}}{\bowtie} ProbeS))) \underset{\mathcal{T}}{\bowtie} Control$$

Communicating Probes Example: big and small servers

```
Small      = (request , ⊤) . SmallComp
           + (break, r) . SmallBroken ;
SmallComp    = (compute , compRate ) . SmallResp ;
SmallResp   = (response, responseRate) . Small ;
SmallBroken = (repair, repairRate) . Small ;

Big     = (request , ⊤) . BigComp
           + (break, r) . BigBroken ;
BigComp    = (compute , 3 * compRate ) . BigResp ;
BigResp   = (response, responseRate) . Big ;
BigBroken = (repair, repairRate) . Big ;

Servers = (Small || Big)
```

$\texttt{Client[4]} \underset{\{request,response\}}{\bowtie} \texttt{Servers}$

Communicating Probes

- Suppose we wish to answer the question "What is the response time if the small server is broken when the request is made"

Communicating Probes

- Suppose we wish to answer the question "What is the response time if the small server is broken when the request is made"
- We do this by adding three separate probes to the model:

Communicating Probes

- Suppose we wish to answer the question "What is the response time if the small server is broken when the request is made"
- We do this by adding three separate probes to the model:
  - $Client :: (work : cwork, response : cresp)$

Communicating Probes

- Suppose we wish to answer the question "What is the response time if the small server is broken when the request is made"
- We do this by adding three separate probes to the model:
  - $Client :: (work : cwork, response : cresp)$
  - $Small :: (break : in, repair : out)$

Communicating Probes

- Suppose we wish to answer the question "What is the response time if the small server is broken when the request is made"
- We do this by adding three separate probes to the model:
  - $Client :: (work : cwork, response : cresp)$
  - $Small :: (break : in, repair : out)$
  - $((in, cwork)/out) : start, cresp : stop)$

Client Probe

```
ClientProbe0 = (work, ⊤) . cwork . ClientProbe1
             + (response, ⊤) . ClientProbe0 ;

ClientProbe1 = (response, ⊤) . cresp . ClientProbe0
             + (work, ⊤) . ClientProbe1 ;
```

Small Probe

```
SmallProbe0 = (break, ⊤) . in . SmallProbe1
            + (repair, ⊤) . SmallProbe0 ;
SmallProbe1 = (repair, ⊤) . out . SmallProbe0
            + (break, ⊤) . SmallProbe1 ;
```

Control Probe

```
Control0 = in . Control1
         + cwork . Control0
         + cresp . Control0
         + out . Control0 ;
Control1 = cwork . start . Control2
         + out . Control0
         + in . Control1
         + cresp . Control1 ;
Control2 = cresp . stop . Control0
         + cwork . Control2
         + in . Control2
         + in . Control2 ;
```

Placing the Local Probes

$$\texttt{PClient} = (\texttt{Client} \underset{\{work,response\}}{\bowtie} \texttt{ClientProbe0})$$

$$\texttt{PSmall} = (\texttt{Small} \underset{\{break,repair\}}{\bowtie} \texttt{SmallProbe0})$$

$$\texttt{Clients} = \texttt{PClient} \bowtie \texttt{Client}[3]$$

$$\texttt{Servers} = \texttt{PSmall} \parallel \texttt{Big}$$

$$\texttt{System} = (\texttt{Clients} \underset{\{request,response\}}{\bowtie} \texttt{Servers}) \underset{\{cwork,cresp,in,out\}}{\bowtie} \texttt{Control0}$$

Refining the Probes

- If we wish to change the question to: "What is the response time if the Big server is broken when the request is made"

Refining the Probes

- If we wish to change the question to: "What is the response time if the Big server is broken when the request is made"
- $Small :: (break : in, repair : out)$

Refining the Probes

- If we wish to change the question to: "What is the response time if the Big server is broken when the request is made"
- $Small :: (break : in, repair : out)$
- $Big :: (break : in, repair : out)$

Refining the Probes

- If we wish to change the question to: "What is the response time if the Big server is broken when the request is made"
- $Small :: (break : in, repair : out)$
- $Big :: (break : in, repair : out)$
- How about, if either of the two servers are broken:

Refining the Probes

- If we wish to change the question to: "What is the response time if the Big server is broken when the request is made"
- $Small :: (break : in, repair : out)$
- $Big :: (break : in, repair : out)$
- How about, if either of the two servers are broken:
- $Servers :: (break : in, (break, repair)*, repair : out)$

Outline

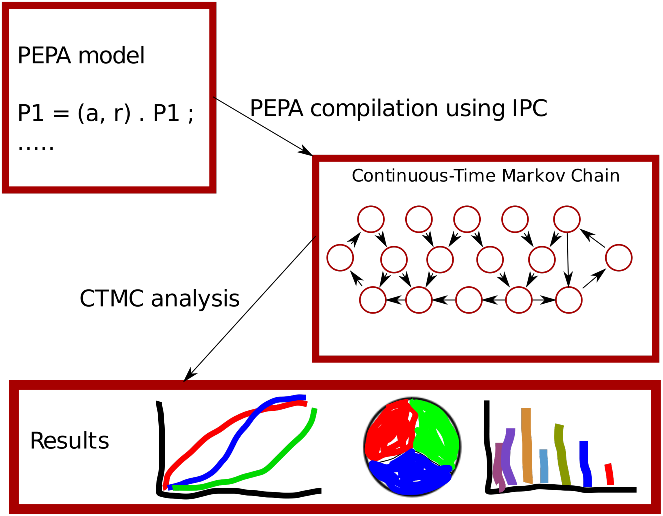
1 Stochastic probes

2 Passage Time and Passage End Analysis

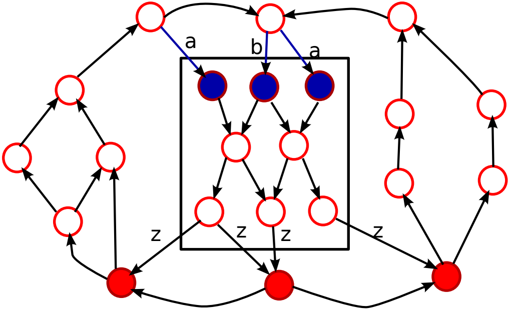
3 Models with a Spatial Aspect
- Spatial Challenge: Capturing physical space

PEPA

We write our models in the PEPA stochastic process algebra



Introduction — Average Response Time



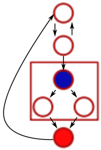
$$\mathit{Response\ Time} = \frac{\text{Probability we are within a passage}}{\text{Throughput of actions which start the passage}}$$

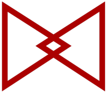
Introduction — Average Response Time

- The above passage may be difficult to specify
- We can describe a passage with a probe
- $(a \mid b) : start, z : stop$
- This is then translated into a PEPA component
- $$\begin{array}{lcl} Probe & \stackrel{def}{=} & (a, \top).Probe_1 \\ & + & (b, \top).Probe_1 \\ Probe_1 & \stackrel{def}{=} & (z, \top).Probe \end{array}$$
- And attached to the main component of the system
- $System \bowtie_{\mathcal{L}} Probe$
  where $\mathcal{L} = \{a, b, z\}$

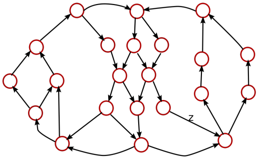
Probes

Probe

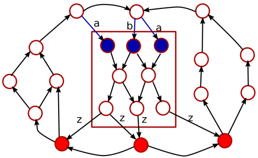




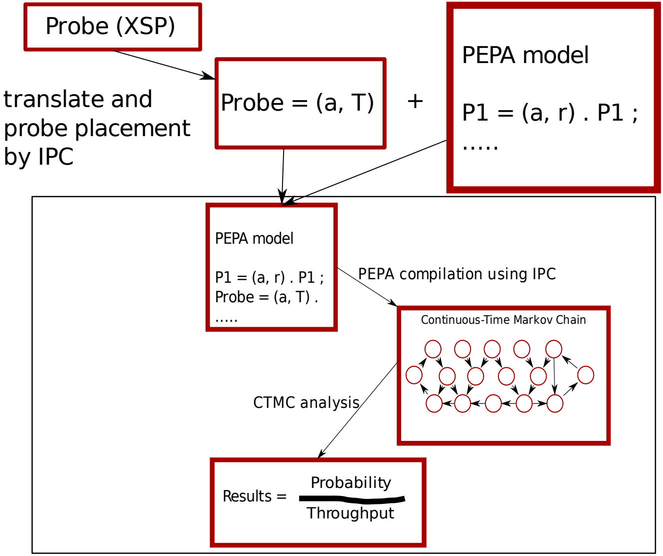
Model



Probed Model



Probe Workflow



Limitations of the average response time

Average response time is quite a crude measure of the performance from a customers perspective.

Limitations of the average response time

Average response time is quite a crude measure of the performance from a customers perspective.

Service Level Agreements are the way that the industry often likes to specify performance requirements and these require a more detailed analysis of responses.

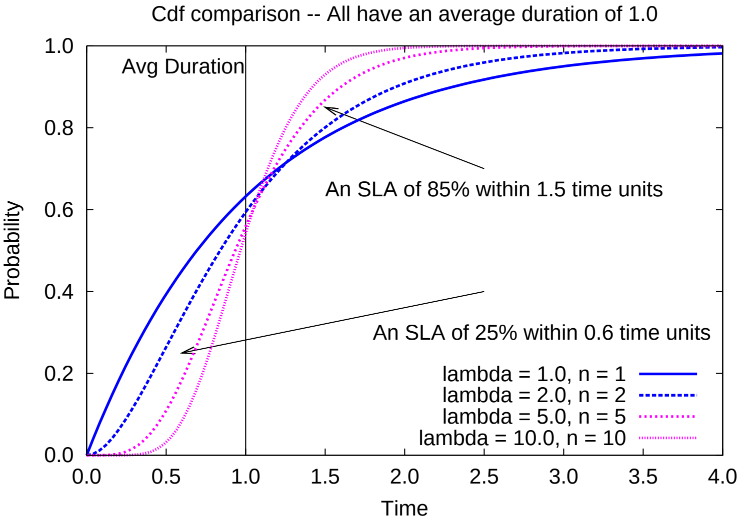
Limitations of the average response time

Average response time is quite a crude measure of the performance from a customers perspective.

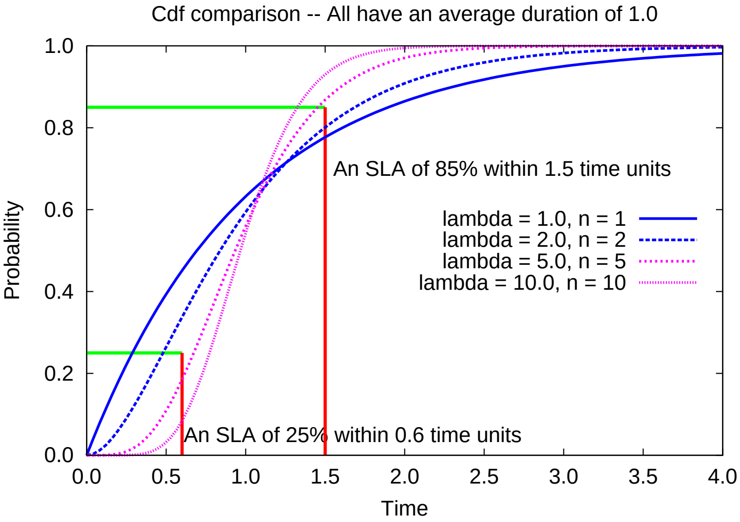
Service Level Agreements are the way that the industry often likes to specify performance requirements and these require a more detailed analysis of responses.

In particular they require the full cumulative (probability) distribution function (cdf) to be calculated for a response time rather than just its average.

Passage times and service level agreements

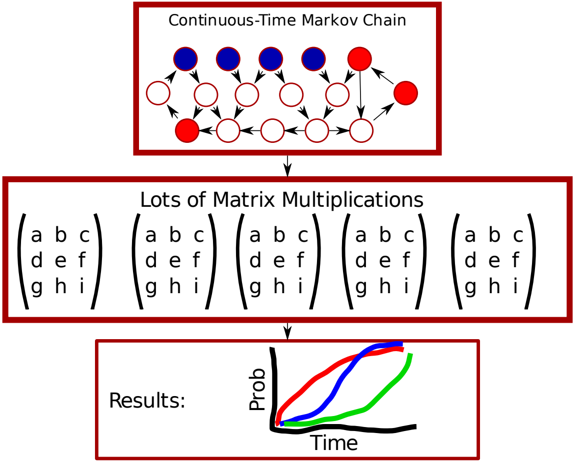


Passage times and service level agreements

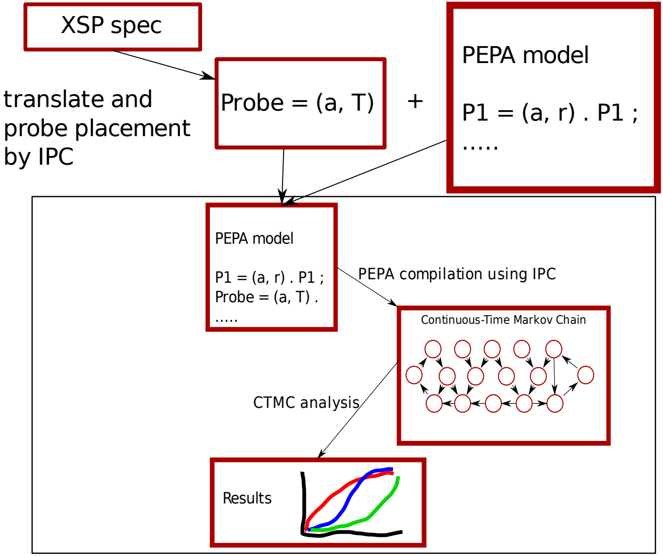


Uniformisation

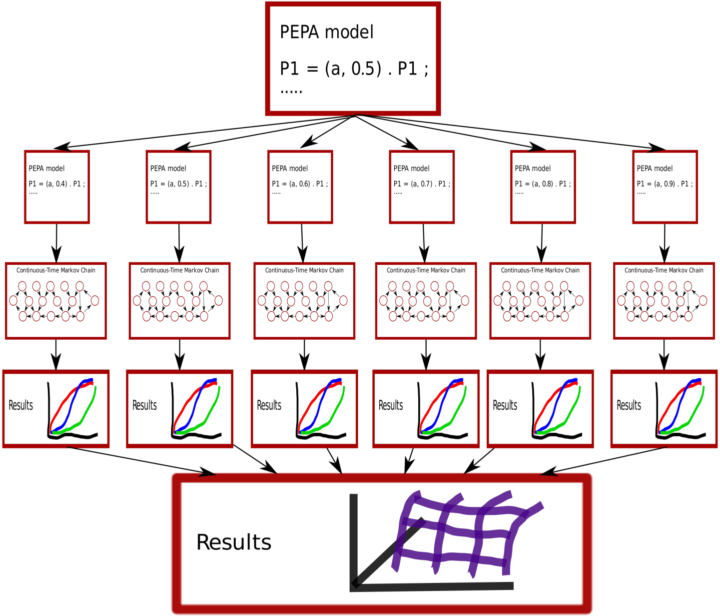
We use a technique called "uniformisation" to calculate the passage-time quantiles or response-time profiles from the CTMC derived from the PEPA model with a suitable probe.



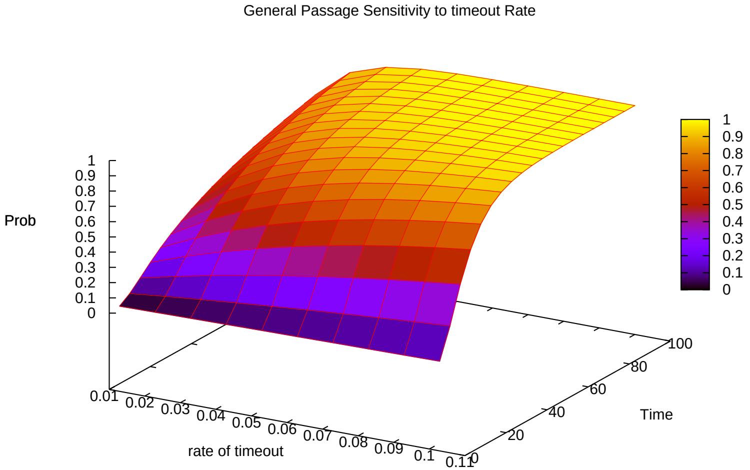
Probe Workflow



Sensitivity Analysis



Sensitivity Analysis



Passage-end Analysis

- However we found that passage-time/response-time analysis was not always expressive enough

Passage-end Analysis

- However we found that passage-time/response-time analysis was not always expressive enough
- In particular when seeking to improve a system to satisfy a particular service level agreement one method is to simply respond negatively more often, or even to every single request

Passage-end Analysis

- However we found that passage-time/response-time analysis was not always expressive enough
- In particular when seeking to improve a system to satisfy a particular service level agreement one method is to simply respond negatively more often, or even to every single request
- For example the naïvely analysed response-time for the financial case study can be improved by simply responding "Loan Application Rejected" to all applications

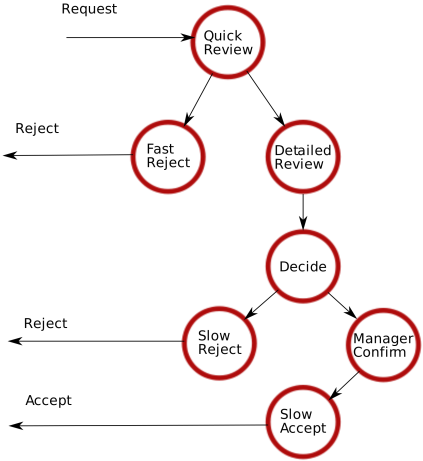
Passage-end Analysis

- However we found that passage-time/response-time analysis was not always expressive enough
- In particular when seeking to improve a system to satisfy a particular service level agreement one method is to simply respond negatively more often, or even to every single request
- For example the naïvely analysed response-time for the financial case study can be improved by simply responding "Loan Application Rejected" to all applications
- Predecide = (approve, p0).App + (decline, p1).Dec + (defer, p2).Def

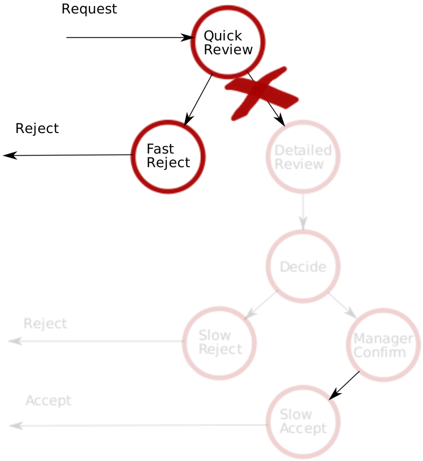
Passage-end Analysis

- However we found that passage-time/response-time analysis was not always expressive enough
- In particular when seeking to improve a system to satisfy a particular service level agreement one method is to simply respond negatively more often, or even to every single request
- For example the naïvely analysed response-time for the financial case study can be improved by simply responding "Loan Application Rejected" to all applications
- Predecide = (approve, p0).App + (decline, p1).Dec + (defer, p2).Def
- Predecide = (decline, p0 + p1 + p2).Dec

Passage-end Analysis



Passage-end Analysis



Passage-end Analysis

- In addition non-expert modellers reported difficulty in constructing queries for response-times with multiple conclusions of which only a subset are to be analysed.

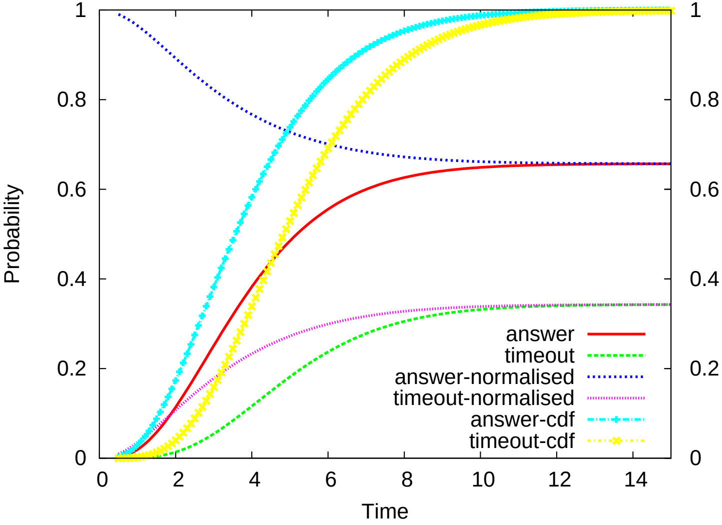
Passage-end Analysis

- In addition non-expert modellers reported difficulty in constructing queries for response-times with multiple conclusions of which only a subset are to be analysed.
- Worse still a simplistic attempt to do so can give back incorrect but not obviously incorrect results (the saving grace being that such results would underestimate system performance).

Passage-end Analysis

- In addition non-expert modellers reported difficulty in constructing queries for response-times with multiple conclusions of which only a subset are to be analysed.
- Worse still a simplistic attempt to do so can give back incorrect but not obviously incorrect results (the saving grace being that such results would underestimate system performance).
- To address these two problems we developed passage-end analysis.

Airbag Example: Passage-end Analysis



Passage-end Analysis: example queries

- Ninety percent of requests are responded to within ten seconds

Passage-end Analysis: example queries

- Ninety percent of requests are responded to within ten seconds
- Ninety percent of requests are responded to within ten seconds and eighty percent of all requests are responded to positively

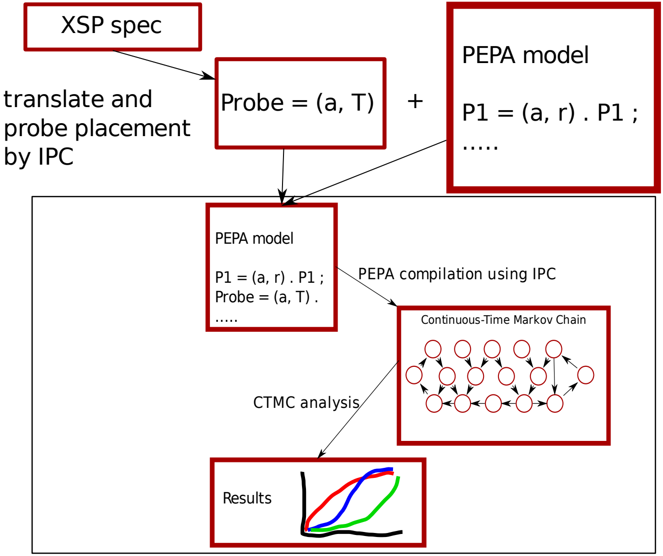
Passage-end Analysis: example queries

- Ninety percent of requests are responded to within ten seconds
- Ninety percent of requests are responded to within ten seconds and eighty percent of all requests are responded to positively
- Ninety percent of requests are responded to within ten seconds and of those eighty percent are positive responses

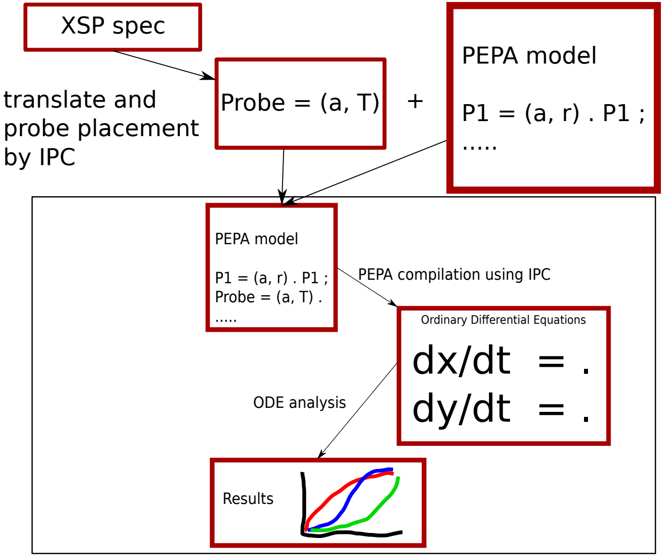
ODE probes

- However the (relevant) analyses possible were limited to average response-time
- During the course of the SENSORIA project we have used the expressiveness of the language of XSP to develop a technique to extract response-time profiles from models which are translated into ODEs
- Additionally because this uses XSP the workflow observed by the user is the same as that when performing analysis via CTMC
- In particular this is due to XSP specifications being translated into PEPA components which when added to the original model result in what is still a PEPA model.

Probe Workflow



Probe Workflow



Another Example — Processor/Resource Model

$$\begin{aligned} Processor_1 &\stackrel{def}{=} (think, r_{think}).Processor_2 \\ Processor_2 &\stackrel{def}{=} (use, r_{use}).Processor_1 \end{aligned}$$

$$\begin{aligned} Resource_1 &\stackrel{def}{=} (use, r_{use}).Resource_2 \\ Resource_2 &\stackrel{def}{=} (reset, r_{reset}).Resource_1 \end{aligned}$$

$$(Processor_1[200]) \underset{\{use\}}{\bowtie} (Resource_1[120])$$

Another Example — Processor/Resource Model

$$
\begin{aligned}
Processor_1 &\stackrel{def}{=} (think, r_{think}).Processor_2 \\
Processor_2 &\stackrel{def}{=} (use, r_{use}).Processor_1
\end{aligned}
$$

$$
\begin{aligned}
Resource_1 &\stackrel{def}{=} (use, r_{use}).Resource_2 \\
Resource_2 &\stackrel{def}{=} (reset, r_{reset}).Resource_1
\end{aligned}
$$

$$(Processor_1[200]) \underset{\{use\}}{\bowtie} (Resource_1[120])$$

From the steady-state analysis:

$$
\begin{aligned}
RunningProcessor_2 &= 111.75 \\
StoppedProcessor_1 &= 88.25
\end{aligned}
$$

Processor - Resource CTMC Response Times

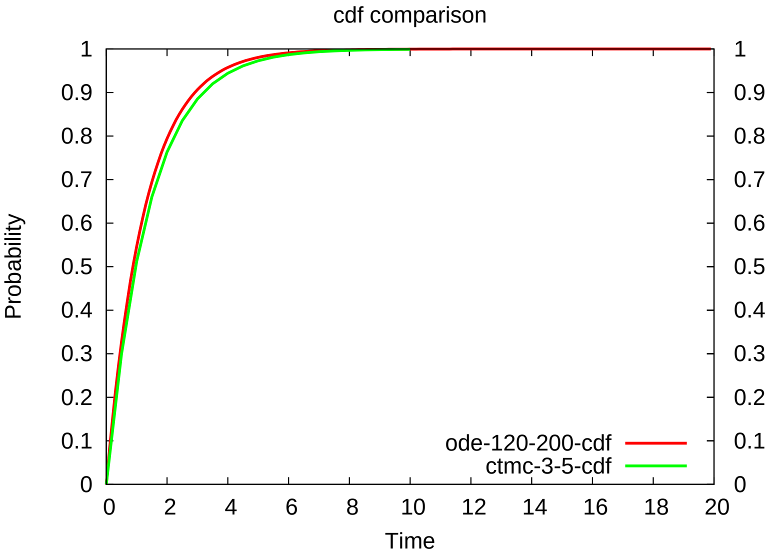
$$Processor_1 \stackrel{def}{=} (think, r_{think}).Processor_2$$
$$Processor_2 \stackrel{def}{=} (use, r_{use}).Processor_1$$

$$Resource_1 \stackrel{def}{=} (use, r_{use}).Resource_2$$
$$Resource_2 \stackrel{def}{=} (reset, r_{reset}).Resource_1$$

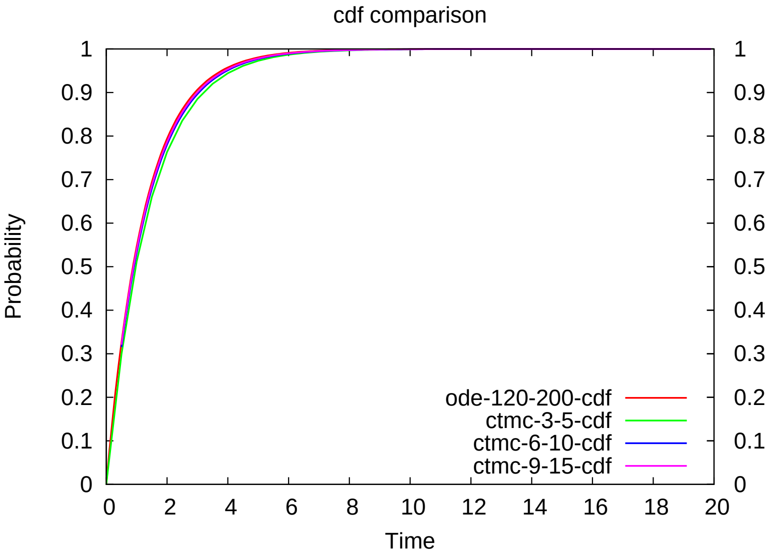
$$(Processor_1[5]) \underset{\{use\}}{\bowtie} (Resource_1[3])$$

$$Probe = Processor_1 :: think : start, use : stop$$

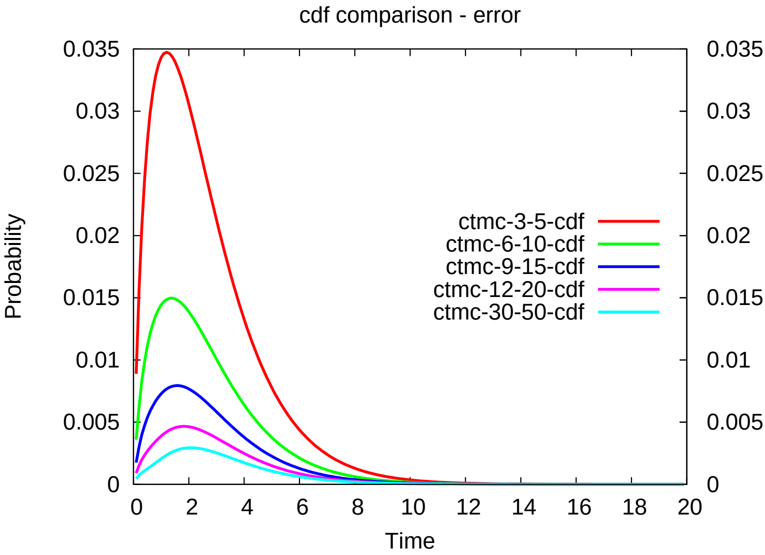
Processor - Resource Comparison



Processor - Resource Comparison



Processor - Resource Comparison



Outline

1 Stochastic probes

2 Passage Time and Passage End Analysis

3 Models with a Spatial Aspect

- Spatial Challenge: Capturing physical space

Spatial Challenge: capturing logical space

Whilst stochastic process algebras are well-suited to model concurrent systems, there is an implicit assumption that all components are co-located.

Spatial Challenge: capturing logical space

Whilst stochastic process algebras are well-suited to model concurrent systems, there is an implicit assumption that all components are co-located.

10-15 years ago we started modelling systems which broke this assumption and demanded more careful thought about the location of components, and how location influences the dynamic evolution of the system.

Spatial Challenge: capturing logical space

Whilst stochastic process algebras are well-suited to model concurrent systems, there is an implicit assumption that all components are co-located.

10-15 years ago we started modelling systems which broke this assumption and demanded more careful thought about the location of components, and how location influences the dynamic evolution of the system.

Mobile devices and mobile computation

The location of components of a software system can have dramatic effect on the performance, particularly as communication is often slow compared with computation. Thus capturing whether components are co-located or communicating over a distance became important.

Spatial Challenge: capturing logical space

Whilst stochastic process algebras are well-suited to model concurrent systems, there is an implicit assumption that all components are co-located.

10-15 years ago we started modelling systems which broke this assumption and demanded more careful thought about the location of components, and how location influences the dynamic evolution of the system.

Biochemical signalling pathways
Far from being a well-mixed soup, the inside of a cell is highly structured and divided into distinct compartments. This physical organisation can have a strong impact on the dynamic behaviour.

Mobile computation: PEPA nets

- The PEPA nets formalism uses the stochastic process algebra PEPA as the inscription language for coloured Petri nets.

Mobile computation: PEPA nets

- The PEPA nets formalism uses the stochastic process algebra PEPA as the inscription language for coloured Petri nets.
- The combination naturally represents applications with two classes of change of state (global and local).

Mobile computation: PEPA nets

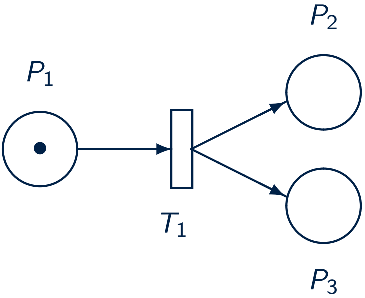
- The PEPA nets formalism uses the stochastic process algebra PEPA as the inscription language for coloured Petri nets.
- The combination naturally represents applications with two classes of change of state (global and local).
- For example, in a mobile code system PEPA terms are used to model the program code which moves between network hosts (the places in the net).

Petri nets

- Petri nets provide a graphical presentation of a model which has an easily accessible interpretation and like process algebras they are supported by an unambiguous formal interpretation.

Petri nets

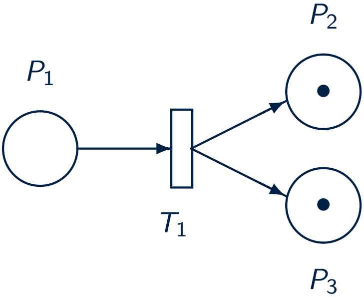
- Petri nets provide a graphical presentation of a model which has an easily accessible interpretation and like process algebras they are supported by an unambiguous formal interpretation.



When a transition fires tokens from input places are absorbed and tokens are created on each of the output places.

Petri nets

- Petri nets provide a graphical presentation of a model which has an easily accessible interpretation and like process algebras they are supported by an unambiguous formal interpretation.



When a transition fires tokens from input places are absorbed and tokens are created on each of the output places.

Petri nets

- Petri nets provide a graphical presentation of a model which has an easily accessible interpretation and like process algebras they are supported by an unambiguous formal interpretation.

Petri nets

- Petri nets provide a graphical presentation of a model which has an easily accessible interpretation and like process algebras they are supported by an unambiguous formal interpretation.
- Coloured Petri nets are a high-level form of classical Petri nets. The plain (indistinguishable) tokens of a classical Petri net are replaced by arbitrary terms which are distinguishable.

Petri nets

- Petri nets provide a graphical presentation of a model which has an easily accessible interpretation and like process algebras they are supported by an unambiguous formal interpretation.
- Coloured Petri nets are a high-level form of classical Petri nets. The plain (indistinguishable) tokens of a classical Petri net are replaced by arbitrary terms which are distinguishable.
- In stochastic Petri nets the transitions from one marking to another are associated with a random variable drawn from an exponential distribution.

Petri nets

- Petri nets provide a graphical presentation of a model which has an easily accessible interpretation and like process algebras they are supported by an unambiguous formal interpretation.
- Coloured Petri nets are a high-level form of classical Petri nets. The plain (indistinguishable) tokens of a classical Petri net are replaced by arbitrary terms which are distinguishable.
- In stochastic Petri nets the transitions from one marking to another are associated with a random variable drawn from an exponential distribution.
- PEPA nets are coloured stochastic Petri nets where the colours used as the tokens of the net are PEPA components.

Global and local state changes

- Firings in a PEPA net (at the Petri net level) model macro-step changes of state such as a mobile software agent moving from one network host to another.

Global and local state changes

- Firings in a PEPA net (at the Petri net level) model macro-step changes of state such as a mobile software agent moving from one network host to another.
- A token/PEPA component will move from one place/context to another.

Global and local state changes

- Firings in a PEPA net (at the Petri net level) model macro-step changes of state such as a mobile software agent moving from one network host to another.
- A token/PEPA component will move from one place/context to another.
- Firings have global effect because they involve components at more than one place in the net.

Global and local state changes

- Firings in a PEPA net (at the Petri net level) model macro-step changes of state such as a mobile software agent moving from one network host to another.
- A token/PEPA component will move from one place/context to another.
- Firings have global effect because they involve components at more than one place in the net.
- A transition occurs whenever an action (individual or shared) of a PEPA component can occur.

Global and local state changes

- Firings in a PEPA net (at the Petri net level) model macro-step changes of state such as a mobile software agent moving from one network host to another.
- A token/PEPA component will move from one place/context to another.
- Firings have global effect because they involve components at more than one place in the net.
- A transition occurs whenever an action (individual or shared) of a PEPA component can occur.
- Since only co-located components can cooperate all transitions have local effect because they involve only components at one place in the net.

Example: a mobile agent system

- A roving agent visits three sites. It interacts with static software components at these sites and has two kinds of interactions.

Example: a mobile agent system

- A roving agent visits three sites. It interacts with static software components at these sites and has two kinds of interactions.
- When visiting a site where a network probe is present it interrogates the probe for the data which it has gathered on recent patterns of network traffic.

Example: a mobile agent system

- A roving agent visits three sites. It interacts with static software components at these sites and has two kinds of interactions.
- When visiting a site where a network probe is present it interrogates the probe for the data which it has gathered on recent patterns of network traffic.
- When it returns to the central co-ordinating site it dumps the data which it has harvested to the master probe. The master probe performs a computationally expensive statistical analysis of the data.

Example: a mobile agent system

- A roving agent visits three sites. It interacts with static software components at these sites and has two kinds of interactions.
- When visiting a site where a network probe is present it interrogates the probe for the data which it has gathered on recent patterns of network traffic.
- When it returns to the central co-ordinating site it dumps the data which it has harvested to the master probe. The master probe performs a computationally expensive statistical analysis of the data.
- The structure of the system allows this computation to be overlapped with the agent's communication and data gathering.

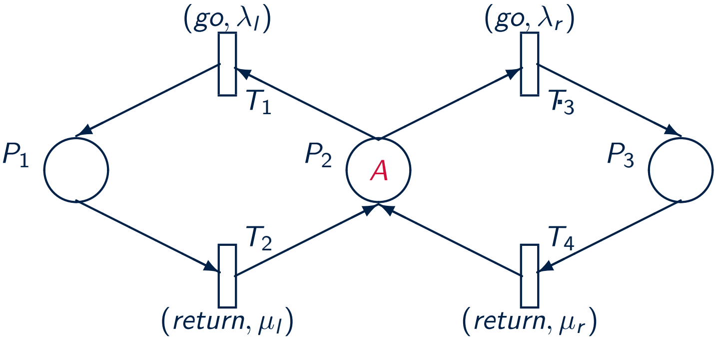
PEPA components

$$
\begin{aligned}
Agent &\stackrel{def}{=} (go, \lambda).Agent' \\
Agent' &\stackrel{def}{=} (interrogate, r_i).Agent'' \\
Agent'' &\stackrel{def}{=} (return, \mu).Agent''' \\
Agent''' &\stackrel{def}{=} (dump, r_d).Agent
\end{aligned}
$$

$$
\begin{aligned}
Master &\stackrel{def}{=} (dump, \top).Master' \\
Master' &\stackrel{def}{=} (analyse, r_a).Master
\end{aligned}
$$

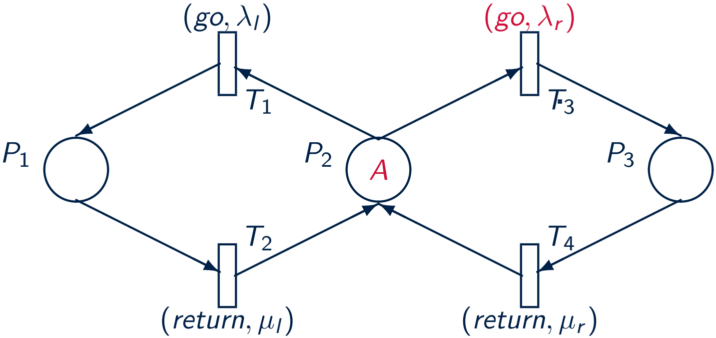
$$
\begin{aligned}
Probe &\stackrel{def}{=} (monitor, r_m).Probe + \\
&\quad\; (interrogate, \top).Probe
\end{aligned}
$$

PEPA net example



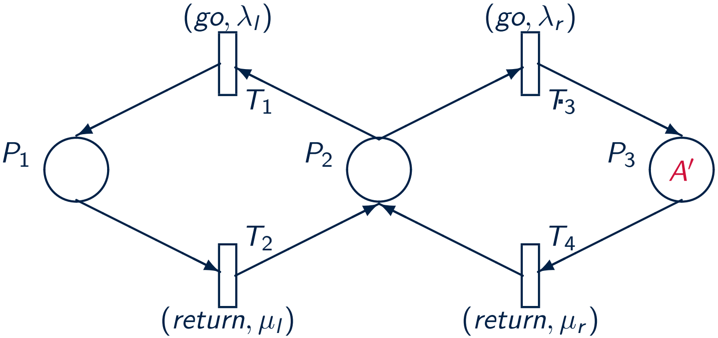
In this model there is a *Master* component located in place P_2, and a *Probe* component located in each of the places P_1 and P_3.

PEPA net example



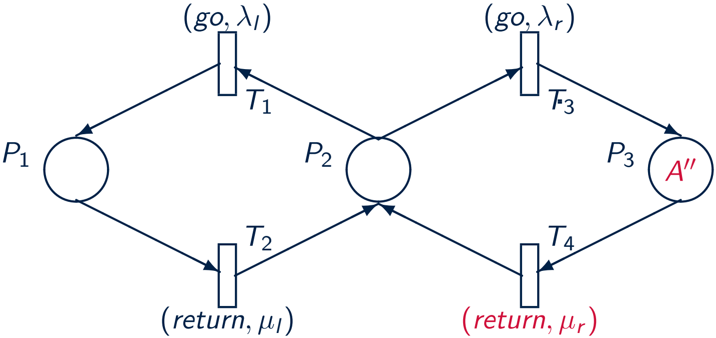
$$
\begin{aligned}
Agent &\stackrel{def}{=} (go, \lambda).Agent' \\
Agent' &\stackrel{def}{=} (interrogate, r_i).Agent'' \\
Agent'' &\stackrel{def}{=} (return, \mu).Agent''' \\
Agent''' &\stackrel{def}{=} (dump, r_d).Agent
\end{aligned}
$$

PEPA net example



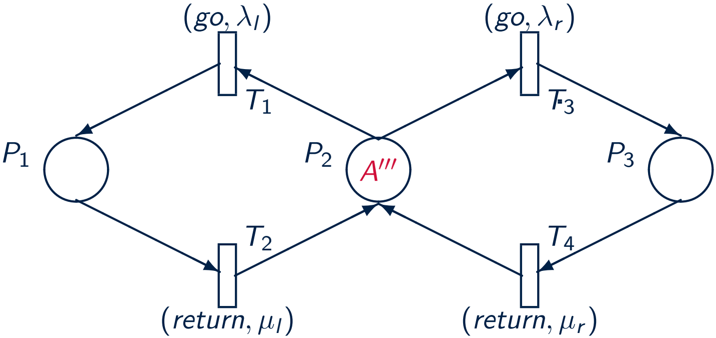
$$\begin{aligned} Agent &\stackrel{def}{=} (go, \lambda).Agent' \\ Agent' &\stackrel{def}{=} (interrogate, r_i).Agent'' \\ Agent'' &\stackrel{def}{=} (return, \mu).Agent''' \\ Agent''' &\stackrel{def}{=} (dump, r_d).Agent \end{aligned}$$

PEPA net example



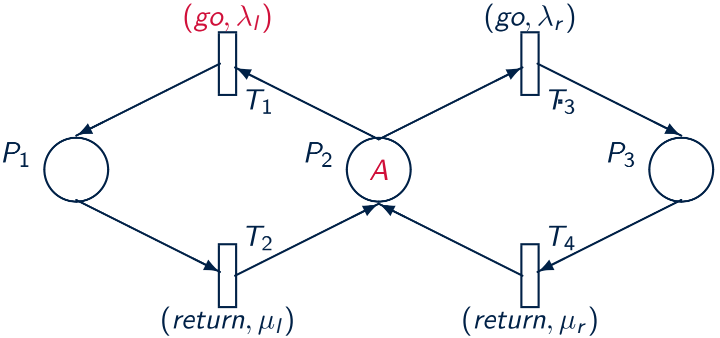
$$
\begin{aligned}
Agent &\stackrel{def}{=} (go, \lambda).Agent' \\
Agent' &\stackrel{def}{=} (interrogate, r_i).Agent'' \\
Agent'' &\stackrel{def}{=} (return, \mu).Agent''' \\
Agent''' &\stackrel{def}{=} (dump, r_d).Agent
\end{aligned}
$$

PEPA net example



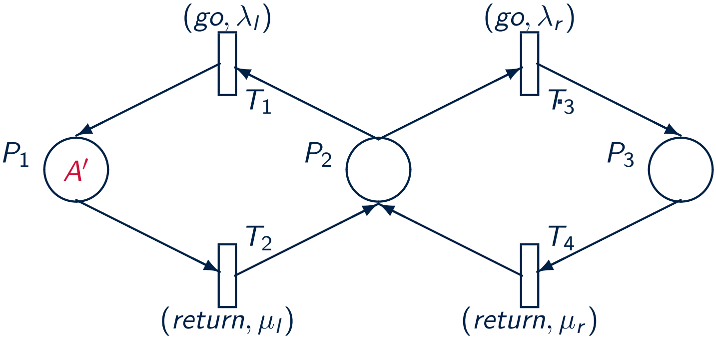
$$
\begin{aligned}
Agent &\stackrel{def}{=} (go, \lambda).Agent' \\
Agent' &\stackrel{def}{=} (interrogate, r_i).Agent'' \\
Agent'' &\stackrel{def}{=} (return, \mu).Agent''' \\
Agent''' &\stackrel{def}{=} (dump, r_d).Agent
\end{aligned}
$$

PEPA net example



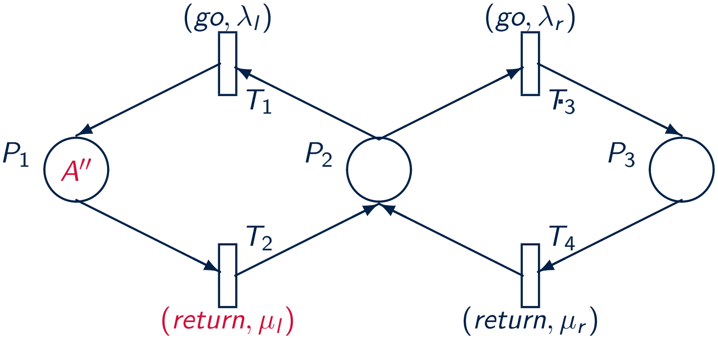
$$
\begin{aligned}
Agent &\stackrel{def}{=} (go, \lambda).Agent' \\
Agent' &\stackrel{def}{=} (interrogate, r_i).Agent'' \\
Agent'' &\stackrel{def}{=} (return, \mu).Agent''' \\
Agent''' &\stackrel{def}{=} (dump, r_d).Agent
\end{aligned}
$$

PEPA net example



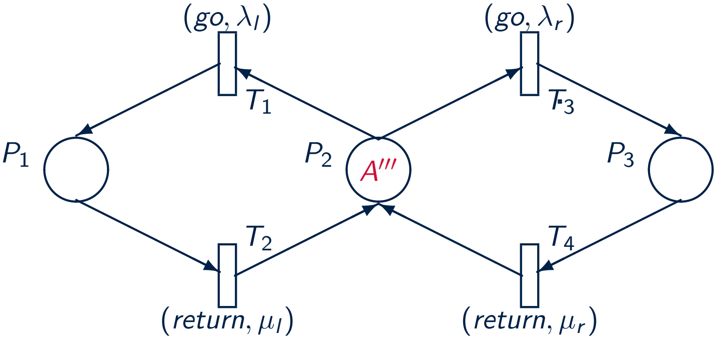
$$
\begin{aligned}
Agent &\stackrel{def}{=} (go, \lambda).Agent' \\
Agent' &\stackrel{def}{=} (interrogate, r_i).Agent'' \\
Agent'' &\stackrel{def}{=} (return, \mu).Agent''' \\
Agent''' &\stackrel{def}{=} (dump, r_d).Agent
\end{aligned}
$$

PEPA net example



$$\begin{aligned}
Agent &\stackrel{def}{=} (go, \lambda).Agent' \\
Agent' &\stackrel{def}{=} (interrogate, r_i).Agent'' \\
Agent'' &\stackrel{def}{=} (return, \mu).Agent''' \\
Agent''' &\stackrel{def}{=} (dump, r_d).Agent
\end{aligned}$$

PEPA net example



$$
\begin{aligned}
Agent &\stackrel{def}{=} (go, \lambda).Agent' \\
Agent' &\stackrel{def}{=} (interrogate, r_i).Agent'' \\
Agent'' &\stackrel{def}{=} (return, \mu).Agent''' \\
Agent''' &\stackrel{def}{=} (dump, r_d).Agent
\end{aligned}
$$

Bio-PEPA

Bio-PEPA is a stochastic process algebra closely related to PEPA, but specifically designed for capturing biochemical network and systems with large interacting populations.

Bio-PEPA

Bio-PEPA is a stochastic process algebra closely related to PEPA, but specifically designed for capturing biochemical network and systems with large interacting populations.

The language contains some constructs to model locations, and particularly pathways involving multiple compartments.

Modelling biological locations

Bio-PEPA considers locations which can be either compartments or membranes.

Modelling biological locations

Bio-PEPA considers locations which can be either compartments or membranes.

Reactions can then be considered to be

- internal to one compartment or membrane
- involving elements in one compartment and one membrane
- transport between compartments.

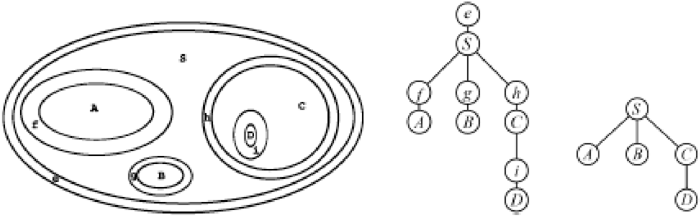
Modelling biological locations

Bio-PEPA considers locations which can be either compartments or membranes.

Reactions can then be considered to be

- internal to one compartment or membrane
- involving elements in one compartment and one membrane
- transport between compartments.

A location tree is used to represent the hierarchy of locations.



Locations in Bio-PEPA

Components in Bio-PEPA are known as species, and in essence, a species in a different location is treated as a distinct species.

However to ease the representation of models, high-level syntax allows some compact representations e.g.

$S \stackrel{def}{=} (\gamma[L_1 \to L_2], \kappa) \odot S$ for transport from location L_1 to location

$S \stackrel{def}{=} (\alpha, \kappa) op S@L_1$ for reaction α at location L_1

Analysing models with logical locations

Both PEPA Nets and Bio-PEPA allow logical locations to be captured within a process algebra model.

Analysing models with logical locations

Both PEPA Nets and Bio-PEPA allow logical locations to be captured within a process algebra model.

However, for analysis they currently rely on an expansion that treats each component, at each location, as distinct.

Analysing models with logical locations

Both PEPA Nets and Bio-PEPA allow logical locations to be captured within a process algebra model.

However, for analysis they currently rely on an expansion that treats each component, at each location, as distinct.

This exacerbates the problem of state space explosion and can limit the size of models that can be analysed.

Analysing models with logical locations

Both PEPA Nets and Bio-PEPA allow logical locations to be captured within a process algebra model.

However, for analysis they currently rely on an expansion that treats each component, at each location, as distinct.

This exacerbates the problem of state space explosion and can limit the size of models that can be analysed.

In particular, fluid approximation techniques can only be used when the population at each location is sufficiently large to justify the continuous approximation.

Moving on to physical space

As we begin to witness informatic environments as Robin Milner defined them, with computational capacity embedded in our physical environment, it is going to become increasingly important to be able to model them and predict their behaviour.

Moving on to physical space

As we begin to witness informatic environments as Robin Milner defined them, with computational capacity embedded in our physical environment, it is going to become increasingly important to be able to model them and predict their behaviour.

In many of these cases, logical location will not be enough and real physical location will need to be incorporated into our modelling techniques.

Moving on to physical space

As we begin to witness informatic environments as Robin Milner defined them, with computational capacity embedded in our physical environment, it is going to become increasingly important to be able to model them and predict their behaviour.

In many of these cases, logical location will not be enough and real physical location will need to be incorporated into our modelling techniques.

This poses significant challenges both of model expression and model solution.

QUANTICOL Examples

The most expensive aspect of the Paris bike sharing system is relocating bikes to where they are needed.

In smart grids and sustainable energy production with limited storage capacity the location of production and demand become important.

Hybrid process algebra HYPE

The hybrid process algebra HYPE captures both continuously varying values and discrete changes in behaviour.

Cartesian coordinates can be represented as continuous variables with appropriate functions to capture movement.

Hybrid process algebra HYPE

The hybrid process algebra HYPE captures both continuously varying values and discrete changes in behaviour.

Cartesian coordinates can be represented as continuous variables with appropriate functions to capture movement.

MSc student Cheng Feng used this approach in HYPE to model ZebraNET, a sensor network in which RFID tags are attached to zebras.

Hybrid process algebra HYPE

The hybrid process algebra HYPE captures both continuously varying values and discrete changes in behaviour.

Cartesian coordinates can be represented as continuous variables with appropriate functions to capture movement.

MSc student Cheng Feng used this approach in HYPE to model ZebraNET, a sensor network in which RFID tags are attached to zebras.

Unfortunately based on hybrid simulation only six zebras could be simulated on a standard machine and fluid techniques are not applicable.