Modelling Methods and Tools

Stephen Gilmore
LFCS, Edinburgh

October 31, 2005
Modelling methods and tools

- Modelling uses abstraction to attempt to represent the essential qualities of a system (while leaving behind the inessential ones).
Modelling methods and tools

- Modelling uses abstraction to attempt to represent the essential qualities of a system (while leaving behind the inessential ones).
- We experiment at one step removed because it is not possible, or not practical, to experiment on the system itself.

"All models are wrong" . . . but some are useful.

Stephen Gilmore. LFCS, University of Edinburgh.
Modelling methods and tools

- Modelling uses abstraction to attempt to represent the essential qualities of a system (while leaving behind the inessential ones).
- We experiment at one step removed because it is not possible, or not practical, to experiment on the system itself.
- “All models are wrong” . . . but some are useful.
Much formalisation in computer science deals with verification.
Much formalisation in computer science deals with verification.

Verification also rests on abstraction. An automaton or a boolean program may represent part of a concrete program.
Modelling and verification

- Much formalisation in computer science deals with verification.
- Verification also rests on abstraction. An automaton or a boolean program may represent part of a concrete program.
- Modellers verify properties. Verifiers use models.
Much formalisation in computer science deals with verification.

Verification also rests on abstraction. An automaton or a boolean program may represent part of a concrete program.

Modellers verify properties. Verifiers use models.

The difference between modelling and verification is the difference between a violin and a fiddle.
Applications help in at least two ways.

- People are more favourably disposed towards theoretical work which has applications.
- Examples can help the reader to understand conceptual material.
The purpose of modelling is to allow analysis.

Insights gained from analysis of the model may increase our understanding of the system being modelled.

The abstraction in the model may mask issues in the genuine article.
We model things which we do not understand well enough already.
What to model? (And what not to)

- We model things which we do not understand well enough already.
  - Don’t model a stack.
What to model? (And what not to)

- We model things which we do not understand well enough already.
  - Don’t model a stack.
- Systems need to have sufficient inherent complexity to deserve to be modelled.
What to model? (And what not to)

- We model things which we do not understand well enough already.
  - Don’t model a stack.
- Systems need to have sufficient inherent complexity to deserve to be modelled.
  - Don’t model bubblesort.
What to model? (And what not to)

- We model things which we do not understand well enough already.
  - Don’t model a stack.
- Systems need to have sufficient inherent complexity to deserve to be modelled.
  - Don’t model bubblesort.
- Modelling novel technology is likely to be considered more interesting than modelling well-known technology.
What to model? (And what not to)

- We model things which we do not understand well enough already.
  - Don’t model a stack.

- Systems need to have sufficient inherent complexity to deserve to be modelled.
  - Don’t model bubblesort.

- Modelling novel technology is likely to be considered more interesting than modelling well-known technology.
  - Don’t model the tape drive on the DEC PDP-11/50.
The DEC PDP-11 series
Choose a credible analysis for the problem.
What analysis should I do?

- Choose a credible analysis for the problem.
- Can’t publish a proof that the Internet is deadlock-free.
Choose a credible analysis for the problem.
  - Can’t publish a proof that the Internet is deadlock-free.
- Choose an appropriate analysis method.
What analysis should I do?

- Choose a credible analysis for the problem.
  - Can’t publish a proof that the Internet is deadlock-free.
- Choose an appropriate analysis method.
  - Can’t publish a proof that the universe is deadlock-free.
What analysis should I do?

- Choose a credible analysis for the problem.
  - Can’t publish a proof that the Internet is deadlock-free.
- Choose an appropriate analysis method.
  - Can’t publish a proof that the universe is deadlock-free.
  - Can’t publish a proof that bubblesort exhibits superconductivity.
Bubblesort conducts electricity without the loss of energy
Outline

1. A modelling language
2. A semantics for the modelling language
3. Tools for the modelling language
Outline

1. A modelling language
2. A semantics for the modelling language
3. Tools for the modelling language
PEPA (Performance Evaluation Process Algebra) is a high-level modelling language for distributed systems. It can be used to develop models of existing systems (abstraction) or designs for proposed ones (specification).

PEPA can capture performance information in a process algebra setting. It is a stochastic process algebra.

Background

There are many existing successful formal approaches to performance evaluation of distributed systems including the following:

- Generalised Stochastic Petri Nets;
- Stochastic Reward Networks; and
- Stochastic Automata Networks.

It is the stated aim of the work on stochastic process algebras (SPAs) to learn from the other approaches.
SPAs have strengths in the areas of **semantic definition**, inherent **compositionality** and the existence of important equivalence relations (including **bisimulation**). This relation provides the basis for aggregation of PEPA models.
The components in a PEPA model engage, cooperatively or individually, in *activities*. Each activity has an *action type* which corresponds to the actions of the system being modelled.
The components in a PEPA model engage, cooperatively or individually, in activities. Each activity has an action type which corresponds to the actions of the system being modelled.

To represent unimportant or unknown actions there is a distinguished action type, $\tau$. 
Every activity in PEPA has an associated activity rate which may be any positive real number, or the distinguished symbol “⊤”, meaning unspecified, read as ‘top’.
Quantitative aspects

Every activity in PEPA has an associated activity rate which may be any positive real number, or the distinguished symbol “⊤”, meaning unspecified, read as ‘top’.

Components and activities are primitives. PEPA also provides a small set of combinators.
PEPA syntax

\[
S \ ::= \ (\alpha, r).S \quad \text{(prefix)} \\
| S_1 + S_2 \quad \text{(choice)} \\
| X \quad \text{(variable)} \\
C \ ::= \ C_1 \ \L □ \ C_2 \quad \text{(cooperation)} \\
| C / L \quad \text{(hiding)} \\
| S \quad \text{(sequential)}
\]
**PEPA: informal semantics (sequential sublanguage)**

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

The activity \((\alpha, r)\) takes time \(\Delta t\) (drawn from the exponential distribution with parameter \(r\)).

\[S_1 + S_2\]

In this choice either \(S_1\) or \(S_2\) will complete an activity first. The other is discarded.
PEPA: informal semantics (combinators)

\[ C_1 \parallel_L C_2 \]

All activities of \( C_1 \) and \( C_2 \) with types in \( L \) are shared: others remain individual.

**NOTATION:** write \( C_1 \parallel C_2 \) if \( L \) is empty.

\[ C / L \]

Activities of \( C \) with types in \( L \) are hidden (\( \tau \) type activities) to be thought of as internal delays.
Example: M/M/1/N/N queue

\[ \begin{align*}
\text{Arrival}_0 & \quad \stackrel{\text{def}}{=} (\text{accept, } \lambda).\text{Arrival}_1 \\
\text{Arrival}_i & \quad \stackrel{\text{def}}{=} (\text{accept, } \lambda).\text{Arrival}_{i+1} + (\text{serve, } \top).\text{Arrival}_{i-1} \\
\text{Arrival}_N & \quad \stackrel{\text{def}}{=} (\text{serve, } \top).\text{Arrival}_{N-1} \\
\text{Server} & \quad \stackrel{\text{def}}{=} (\text{serve, } \mu).\text{Server}
\end{align*} \]
Example: M/M/1/N/N queue

\[\text{Queue}_i \equiv \text{Arrival}_i \{\text{serve}\} \text{ Server}\]
Expansion Law

\[ P \oplus_L Q = \]
Expansion Law

\[ P \overset{L}{\bowtie} Q = \]

\[ \sum\{ (\alpha, r). (P' \overset{L}{\bowtie} Q) : P \xrightarrow{(\alpha, r)} P'; \alpha \notin L \} + \]
Expansion Law

\[
P \otimes_L Q = \sum\{ (\alpha, r).(P' \otimes_L Q): P \xrightarrow{(\alpha, r)} P'; \alpha \notin L \} + \sum\{ (\alpha, r).(P \otimes_L Q'): Q \xrightarrow{(\alpha, r)} Q'; \alpha \notin L \} + \sum\{ (\alpha, r).(P' \otimes_L Q'): Q \xrightarrow{(\alpha, r)} Q'; \alpha \notin L \} + \text{...}
\]
Expansion Law

\[ P \uplus_{L} Q = \]

\[ \sum\{(\alpha, r). (P' \uplus_{L} Q) : P \xrightarrow{\alpha, r} P'; \alpha \notin L\} + \]

\[ \sum\{(\alpha, r). (P \uplus_{L} Q') : Q \xrightarrow{\alpha, r} Q'; \alpha \notin L\} + \]

\[ \sum\{(\alpha, r). (P' \uplus_{L} Q') : P \xrightarrow{\alpha, r_1} P'; Q \xrightarrow{\alpha, r_2} Q'; \alpha \in L\} \]
What should be the impact of synchronisation on rate? There are many possibilities.

- Restrict synchronisations to have one active partner and one passive partner.
Synchronisation

What should be the impact of synchronisation on rate? There are many possibilities.

- Restrict synchronisations to have one active partner and one passive partner.
- Choose a function which satisfies a small number of algebraic properties.
Synchronisation

What should be the impact of synchronisation on rate? There are many possibilities.

- Restrict synchronisations to have one active partner and one passive partner.
- Choose a function which satisfies a small number of algebraic properties.
- Have the rate limited by the slowest participant in terms of apparent rate. This is the approach adopted by PEPA.
Bounded capacity

Within the cooperation framework, PEPA assumes bounded capacity: that is, a component cannot be made to perform an activity faster by cooperation, so the rate of a shared activity is the minimum of the apparent rates of the activity in the cooperating components.
The total capacity of a component $P$ to carry out activities of type $\alpha$ is termed the **apparent rate** of $\alpha$ in $P$, denoted $r_\alpha(P)$.
The total capacity of a component $P$ to carry out activities of type $\alpha$ is termed the **apparent rate** of $\alpha$ in $P$, denoted $r_\alpha(P)$.

It is used heavily when calculating the pairwise cooperation rate: when cooperating with another component, the bounded capacity principle ensures that the overall rate of cooperation does not exceed either of the constituent apparent rates.
To summarise the original ruleset from [Hillston 96], the apparent rate function can be defined as:

\[ r_\alpha(P) = \sum_{P \xrightarrow{(\alpha, \lambda_i)}} \lambda_i \]  

(1)

where \( \lambda_i \in \mathbb{R}^+ \cup \{ nT \mid n \in \mathbb{Q}, n > 0 \} \), \( nT \) is shorthand for \( n \times T \) and \( T \) represents the passive action rate that inherits the rate of the coaction from the cooperating component.
Properties of $\top$ (the “unspecified” symbol)

$\top$ requires the following arithmetic rules:

\[
\begin{align*}
    m\top < n\top & : \text{ for } m < n \text{ and } m, n \in \mathbb{Q} \\
    r < n\top & : \text{ for all } r \in \mathbb{R}, n \in \mathbb{Q} \\
    m\top + n\top = (m + n)\top & : m, n \in \mathbb{Q} \\
    \frac{m\top}{n\top} = \frac{m}{n} & : m, n \in \mathbb{Q}
\end{align*}
\]
Properties of $\top$ (the “unspecified” symbol)

$\top$ requires the following arithmetic rules:

\[
\begin{align*}
    m \top < n \top & : \text{ for } m < n \text{ and } m, n \in \mathbb{Q} \\
    r < n \top & : \text{ for all } r \in \mathbb{R}, n \in \mathbb{Q} \\
    m \top + n \top = (m + n) \top & : m, n \in \mathbb{Q} \\
    \frac{m \top}{n \top} = \frac{m}{n} & : m, n \in \mathbb{Q}
\end{align*}
\]

Note that $(r + n \top)$ is undefined for all $r \in \mathbb{R}$ in PEPA therefore disallowing components which enable both active and passive actions in the same action type at the same time, e.g. $(\alpha, \lambda).P + (\alpha, \top).P'$.
To apply probability theory to the process under study, we view it as a random experiment.
To apply probability theory to the process under study, we view it as a **random experiment**.

The **sample space** of a random experiment is the set of all individual outcomes of the experiment.
To apply probability theory to the process under study, we view it as a random experiment.

The sample space of a random experiment is the set of all individual outcomes of the experiment.

These individual outcomes are also called sample points or elementary events.
To apply probability theory to the process under study, we view it as a random experiment.

The sample space of a random experiment is the set of all individual outcomes of the experiment.

These individual outcomes are also called sample points or elementary events.

An event is a subset of a sample space.
Random variables

We are interested in the dynamics of a system as events happen over time. A function which associates a (real-valued) number with the outcome of an experiment is known as a **random variable**. Formally, a random variable $X$ is a real-valued function defined on a sample space $\Omega$. 
If $X$ is a random variable, and $x$ is a real number, we write $X \leq x$ for the event

$$\{ \omega : \omega \in \Omega \text{ and } X(\omega) \leq x \}$$
If $X$ is a random variable, and $x$ is a real number, we write $X \leq x$ for the event

$$\{ \omega : \omega \in \Omega \text{ and } X(\omega) \leq x \}$$

and we write $X = x$ for the event

$$\{ \omega : \omega \in \Omega \text{ and } X(\omega) = x \}$$
If $X$ is a random variable, and $x$ is a real number, we write $X \leq x$ for the event

$$\{ \omega : \omega \in \Omega \text{ and } X(\omega) \leq x \}$$

and we write $X = x$ for the event

$$\{ \omega : \omega \in \Omega \text{ and } X(\omega) = x \}$$

Another property required of a random variable is that the set $X \leq x$ is an event for each real $x$. This is necessary so that probability calculations can be made. A function having this property is said to be a measurable function or measurable in the Borel sense.
For each random variable $X$ we define its distribution function $F$ for each real $x$ by

$$F(x) = \Pr[X \leq x]$$
Distribution function

For each random variable $X$ we define its distribution function $F$ for each real $x$ by

$$F(x) = \Pr[X \leq x]$$

We associate another function $p(\cdot)$, called the probability mass function of $X$ (pmf), for each real $x$:

$$p(x) = \Pr[X = x]$$
A random variable $X$ is continuous if $p(x) = 0$ for all real $x$. 

These notes are from Stephen Gilmore, LFCS, University of Edinburgh.
A random variable $X$ is **continuous** if $p(x) = 0$ for all real $x$.

(If $X$ is a continuous random variable, then $X$ can assume infinitely many values, and so it is reasonable that the probability of its assuming any specific value we choose beforehand is zero.)
A random variable $X$ is **continuous** if $p(x) = 0$ for all real $x$.

(If $X$ is a *continuous* random variable, then $X$ can assume infinitely many values, and so it is reasonable that the probability of its assuming any *specific* value we choose beforehand is zero.)

The distribution function for a continuous random variable is a continuous function in the usual sense.
The random variable $X$ is said to be an \textit{exponential random variable with parameter $\lambda$} ($\lambda > 0$) or to have an \textit{exponential distribution with parameter $\lambda$} if it has the distribution function

\[ F(x) = \begin{cases} 1 - e^{-\lambda x} & \text{for } x > 0 \\ 0 & \text{for } x \leq 0 \end{cases} \]
The random variable $X$ is said to be an \textit{exponential random variable} with parameter $\lambda$ ($\lambda > 0$) or to have an \textit{exponential distribution} with parameter $\lambda$ if it has the distribution function

$$F(x) = \begin{cases} 1 - e^{-\lambda x} & \text{for } x > 0 \\ 0 & \text{for } x \leq 0 \end{cases}$$

Some authors call this distribution the \textit{negative exponential distribution}. 
The density function $f = dF/dx$ is given by

$$f(x) = \begin{cases} 
\lambda e^{-\lambda x} & \text{if } x > 0 \\
0 & \text{if } x \leq 0
\end{cases}$$
We sometimes instead see functions such as

\[ f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \]

written as

\[ f(x) = \lambda e^{-\lambda x} 1_{x>0} \]

The function \( 1_{x>0} \) is an **indicator function** (used to code the conditional part of the definition). A Computer Scientist would write this as \( \text{if } x>0 \text{ then } 1 \text{ else } 0 \) or \((x > 0) ? 1 : 0\).
If $X$ is a continuous random variable with density function $f(\cdot)$, we define the mean or expected value of $X$, $\mu = E[X]$ by

$$\mu = E[X] = \int_{-\infty}^{\infty} xf(x)\,dx$$
Suppose $X$ has an exponential distribution with parameter $\lambda > 0$. Then

$$\mu = E[X] = \int_{-\infty}^{\infty} x \lambda e^{-\lambda x} \, dx$$
Suppose $X$ has an exponential distribution with parameter $\lambda > 0$. Then

$$\mu = E[X] = \int_{-\infty}^{\infty} x \lambda e^{-\lambda x} dx = \frac{1}{\lambda}$$
Exponential inter-event time distribution

The time interval between successive events can also be deduced. Let $F(t)$ be the distribution function of $T$, the time between events. Consider $\Pr(T > t) = 1 - F(t)$:

$$\Pr(T > t) = \Pr(\text{No events in an interval of length } t)$$
The time interval between successive events can also be deduced. Let $F(t)$ be the distribution function of $T$, the time between events. Consider $\Pr(T > t) = 1 - F(t)$:

$$\Pr(T > t) = \Pr(\text{No events in an interval of length } t) = 1 - F(t)$$
The time interval between successive events can also be deduced. Let $F(t)$ be the distribution function of $T$, the time between events. Consider $Pr(T > t) = 1 - F(t)$:

$$Pr(T > t) = Pr(\text{No events in an interval of length } t)$$
$$= 1 - F(t)$$
$$= 1 - (1 - e^{-\lambda t})$$
Exponential inter-event time distribution

The time interval between successive events can also be deduced. Let $F(t)$ be the distribution function of $T$, the time between events. Consider $Pr(T > t) = 1 - F(t)$:

\[
Pr(T > t) = Pr(No \text{ events in an interval of length } t) = 1 - F(t) = 1 - (1 - e^{-\lambda t}) = e^{-\lambda t}
\]
The memoryless property of the exponential distribution is so called because the time to the next event is independent of when the last event occurred.
Memoryless property of the exponential distribution

Suppose that the last event was at time 0. What is the probability that the next event will be after $t + s$, given that time $t$ has elapsed since the last event, and no events have occurred?

$$\Pr(T > t + s | T > t) = \Pr(T > t + s \text{ and } T > t) \Pr(T > t)$$

$$= e^{-\lambda(t + s)} e^{-\lambda t}$$

$$= e^{-\lambda s}$$

This value is independent of $t$ (and so the time already spent has not been remembered).
Memoryless property of the exponential distribution

Suppose that the last event was at time 0. What is the probability that the next event will be after $t + s$, given that time $t$ has elapsed since the last event, and no events have occurred?

$$
Pr(T > t + s \mid T > t) = \frac{Pr(T > t + s \text{ and } T > t)}{Pr(T > t)}
$$

This value is independent of $t$ (and so the time already spent has not been remembered).
Suppose that the last event was at time 0. What is the probability that the next event will be after \( t + s \), given that time \( t \) has elapsed since the last event, and no events have occurred?

\[
\Pr(T > t + s \mid T > t) = \frac{\Pr(T > t + s \text{ and } T > t)}{\Pr(T > t)}
\]

\[
= \frac{e^{-\lambda(t+s)}}{e^{-\lambda t}}
\]

This value is independent of \( t \) (and so the time already spent has not been remembered).
Memoryless property of the exponential distribution

Suppose that the last event was at time 0. What is the probability that the next event will be after \( t + s \), given that time \( t \) has elapsed since the last event, and no events have occurred?

\[
\Pr( T > t + s \mid T > t ) = \frac{ \Pr( T > t + s \text{ and } T > t ) }{ \Pr( T > t ) }
\]

\[
= \frac{ e^{-\lambda(t+s)} }{ e^{-\lambda t} } 
\]

\[
= e^{-\lambda s}
\]
Suppose that the last event was at time 0. What is the probability that the next event will be after $t + s$, given that time $t$ has elapsed since the last event, and no events have occurred?

$$Pr(T > t + s | T > t) = \frac{Pr(T > t + s \text{ and } T > t)}{Pr(T > t)} = \frac{e^{-\lambda(t+s)}}{e^{-\lambda t}} = e^{-\lambda s}$$

This value is independent of $t$ (and so the time already spent has not been remembered).
When enabled an activity, \( a = (\alpha, \lambda) \), will delay for a period determined by its associated distribution function, i.e. the probability that the activity \( a \) happens within a period of time of length \( t \) is \( F_a(t) = 1 - e^{-\lambda t} \).
We can think of this as the activity setting a timer whenever it becomes enabled. The time allocated to the timer is determined by the rate of the activity. If several activities are enabled at the same time each will have its own associated timer. When the first timer finishes that activity takes place—the activity is said to complete or succeed. This means that the activity is considered to “happen”: an external observer will witness the event of an activity of type $\alpha$. An activity may be preempted, or aborted, if another one completes first.
In a PEPA model if we define the stochastic process $X(t)$, such that $X(t) = C_i$ indicates that the system behaves as component $C_i$ at time $t$, then $X(t)$ is a Markov process which can be characterised by a matrix, $Q$. 

A stationary or equilibrium probability distribution, $\pi(\cdot)$, exists for every time-homogeneous irreducible Markov process whose states are all positive-recurrent. This distribution is found by solving the global balance equation $\pi Q = 0$ subject to the normalisation condition $\sum \pi(C_i) = 1$. 

Stephen Gilmore. LFCS, University of Edinburgh.
In a PEPA model if we define the stochastic process $X(t)$, such that $X(t) = C_i$ indicates that the system behaves as component $C_i$ at time $t$, then $X(t)$ is a **Markov process** which can be characterised by a matrix, $Q$.

A **stationary** or **equilibrium** probability distribution, $\pi(\cdot)$, exists for every time-homogeneous irreducible Markov process whose states are all positive-recurrent.
PEPA and Markov processes

In a PEPA model if we define the stochastic process $X(t)$, such that $X(t) = C_i$ indicates that the system behaves as component $C_i$ at time $t$, then $X(t)$ is a Markov process which can be characterised by a matrix, $Q$.

A stationary or equilibrium probability distribution, $\pi(\cdot)$, exists for every time-homogeneous irreducible Markov process whose states are all positive-recurrent.

This distribution is found by solving the global balance equation

$$\pi Q = 0$$

subject to the normalisation condition

$$\sum \pi(C_i) = 1.$$
All PEPA models are time-homogeneous since all activities are time-homogeneous: the rate and type of activities enabled by a component are independent of time.
The other conditions, irreducibility and positive-recurrent states, are easily expressed in terms of the derivation graph of the PEPA model. We only consider PEPA models with a finite number of states so if the model is irreducible then all states must be positive-recurrent i.e. the derivation graph is strongly connected.
In terms of the PEPA model this means that all behaviours of the system must be recurrent; in particular, for every choice, whichever path is chosen it must eventually return to the point where the choice can be made again, possibly with a different outcome.
A Markov process with discrete state space and discrete index set is called a Markov chain. The future behaviour of a Markov chain depends only on its current state, and not on how that state was reached. This is the Markov, or memoryless, property.

\[
\Pr(X(t_{n+1}) = x_{n+1} \mid X(t_n) = x_n, \ldots, X(t_0) = x_0) = \Pr(X(t_{n+1}) = x_{n+1} \mid X(t_n) = x_n)
\]
Structured Operational Semantics

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

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

Prefix

$((\alpha, r) . E) \xrightarrow{(\alpha, r)} E$
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

\[
E \xrightarrow{(\alpha,r)} E' \\
E \otimes L F \xrightarrow{(\alpha,r)} E' \otimes L F
\]
Cooperation

\[ E \quad (\alpha, r) \quad E' \]
\[ E \bowtie L \quad F \quad (\alpha, r) \quad E' \bowtie L \quad F \]

\[ F \quad (\alpha, r) \quad F' \]
\[ E \bowtie L \quad F \quad (\alpha, r) \quad E \bowtie L \quad F' \]
Structured Operational Semantics: Cooperation ($\alpha \in L$)

Cooperation

\[ E \xrightarrow{\alpha, r_1} E' \quad F \xrightarrow{\alpha, r_2} F' \]

\[ E \bowtie_L F \xrightarrow{\alpha, R} E' \bowtie_L F' \]
Structured Operational Semantics: Cooperation ($\alpha \in L$)

Cooperation

\[
\begin{align*}
E \xrightarrow{(\alpha,r_1)} & E' \\
F \xrightarrow{(\alpha,r_2)} & F' \\
E \triangleright L \ F & \xrightarrow{(\alpha,R)} \ E' \triangleright L \ F'
\end{align*}
\]

where 

\[
R = \frac{r_1}{r_\alpha(E)} \frac{r_2}{r_\alpha(F)} \min(r_\alpha(E), r_\alpha(F))
\]
Structured Operational Semantics: Hiding

Hiding

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

Hiding

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

(\alpha \notin L)

(\alpha \in L)
Constant

\[
\begin{align*}
E \xrightarrow{(\alpha,r)} E' & \quad (A \overset{\text{def}}{=} E) \\
A \xrightarrow{(\alpha,r)} E' & \end{align*}
\]
PEPA has no “nil” (a deadlocked process). This is because the PEPA language is intended for modelling non-stop processes (such as Web servers, operating systems, or manufacturing processes) rather than for modelling terminating processes (a compilation, a sorting routine, and so forth).
Roll your own!

When we are interested in transient behaviour we use the deadlocked process \( \text{Stop} \) to signal a component which performs no further actions.

\[
\text{Stop} \overset{\text{def}}{=} \left( ((a, r).\text{Stop}) \otimes_{\{a,b\}} ((b, r).\text{Stop}) \right) / \{a, b\}
\]
Cooperation in PEPA is **multi-way**. Two, three, four or more partners may cooperate, and they all need to synchronise for the activity to happen.
Cooperation in PEPA is multi-way. Two, three, four or more partners may cooperate, and they all need to synchronise for the activity to happen.

This comes from the fact that synchronisation has the form $a, a \rightarrow a$ (as in CSP) instead of $a, \bar{a} \rightarrow \tau$ (as in CCS and the $\pi$-calculus).
Cooperation in PEPA is **multi-way**. Two, three, four or more partners may cooperate, and they all need to synchronise for the activity to happen.

This comes from the fact that synchronisation has the form $a, a \rightarrow a$ (as in CSP) instead of $a, \bar{a} \rightarrow \tau$ (as in CCS and the $\pi$-calculus).

This is used to have “witnesses” to events (known as **stochastic probes**).
Because of its mapping onto a CTMC, PEPA has an **interleaving semantics**.
Because of its mapping onto a CTMC, PEPA has an interleaving semantics.

Other modelling formalisms based on CTMCs are also based on an interleaving semantics (e.g. Generalised Stochastic Petri nets).
The importance of being exponential

\[(\alpha, r).\text{Stop} \parallel (\beta, s).\text{Stop}\]

\[
\begin{align*}
(\alpha, r) & \quad (\beta, s) \\
\text{Stop} \parallel (\beta, s).\text{Stop} & \quad (\alpha, r).\text{Stop} \parallel \text{Stop} \\
(\beta, s) & \quad (\alpha, r) \\
\text{Stop} \parallel \text{Stop} & \quad \text{Stop} \parallel \text{Stop}
\end{align*}
\]
The importance of being exponential

\[(\alpha, r).\text{Stop} \parallel (\beta, s).\text{Stop}\]

\[(\alpha, r)\]
\[
\text{Stop} \parallel (\beta, s).\text{Stop}
\]

\[(\beta, s)\]
\[
\text{Stop} \parallel \text{Stop}
\]

\[(\beta, s)\]
\[
(\alpha, r).\text{Stop} \parallel \text{Stop}
\]

\[(\alpha, r)\]
\[
(\alpha, r).\text{Stop} \parallel (\beta, s).\text{Stop}
\]
The importance of being exponential

\[(\alpha, r).\text{Stop} \parallel (\beta, s).\text{Stop}\]

\[\text{Stop} \parallel (\beta, s).\text{Stop} \parallel (\alpha, r).\text{Stop} \parallel \text{Stop}\]

\[\text{Stop} \parallel \text{Stop} \parallel \text{Stop} \parallel (\alpha, r)\]
The importance of being exponential

\[(\alpha, r).\text{Stop} \parallel (\beta, s).\text{Stop}\]

\[(\alpha, r) \quad (\alpha, r) \quad (\beta, s) \quad (\beta, s)\]

\[\text{Stop} \parallel (\beta, s).\text{Stop} \quad (\alpha, r).\text{Stop} \parallel \text{Stop} \quad (\alpha, r) \quad (\beta, s)\]

\[(\beta, s) \quad \text{Stop} \parallel \text{Stop} \quad (\alpha, r)\]
The importance of being exponential

\[(\alpha, r).Stop \parallel (\beta, s).Stop\]

\[(\alpha, r) \rightarrow (\beta, s)\]

\[Stop \parallel (\beta, s).Stop\]

\[(\beta, s) \rightarrow (\alpha, r)\]

\[Stop \parallel Stop\]

\[(\alpha, r).Stop \parallel \text{Stop}\]
The importance of being exponential

\[(\alpha, r).\text{Stop} \parallel (\beta, s).\text{Stop}\]

\[\text{Stop} \parallel (\beta, s).\text{Stop}\]

\[\text{Stop} \parallel \text{Stop}\]

\[\text{Stop} \parallel (\alpha, r).\text{Stop}\]

\[\text{Stop} \parallel (\alpha, r)\]
The importance of being exponential

\[(\alpha, r).Stop \parallel (\beta, s).Stop\]

\[(\alpha, r) \rightarrow \text{Stop} \parallel (\beta, s).\text{Stop}\]

\[(\beta, s) \rightarrow \text{Stop} \parallel (\alpha, r).\text{Stop} \parallel \text{Stop}\]

\[(\beta, s) \rightarrow \text{Stop} \parallel \text{Stop}\]

\[(\alpha, r) \rightarrow \text{Stop} \parallel \text{Stop}\]
The memoryless property of the negative exponential distribution means that residual times do not need to be recorded.
The importance of being exponential

We retain the expansion law of classical process algebra:

\[(\alpha, r).\text{Stop} \parallel (\beta, s).\text{Stop} = (\alpha, r).(\beta, s).\text{Stop} \parallel \text{Stop} + (\beta, s).(\alpha, r).\text{Stop} \parallel \text{Stop}\]

only if the negative exponential distribution is assumed.
A continuous time Markov chain (CTMC) is generated from a PEPA model via its structured operational semantics.
A continuous time Markov chain (CTMC) is generated from a PEPA model via its structured operational semantics.

Linear algebra is used to solve the model in terms of equilibrium behaviour.
A continuous time Markov chain (CTMC) is generated from a PEPA model via its structured operational semantics.

Linear algebra is used to solve the model in terms of equilibrium behaviour.

The resulting probability distribution is seldom the ultimate goal of performance analysis; a modeller derives performance measures from this distribution via a reward structure.
The expression, and testing for satisfaction of equilibrium properties, can be seen to be closely related to the specification, and model checking of a formula expressed in Larsen and Skou’s probabilistic modal logic (PML). We give a modified interpretation of such formulae suitable for reasoning about PEPA’s continuous time models.
The expression, and testing for satisfaction of equilibrium properties, can be seen to be closely related to the specification, and model checking of a formula expressed in Larsen and Skou’s probabilistic modal logic (PML). We give a modified interpretation of such formulae suitable for reasoning about PEPA’s continuous time models.

We exploit the operators of modal logic to be more discriminating about which states contribute to the reward measure. In particular, we can select a state based on model behaviour which is not immediately local to the state.
Larsen and Skou’s PML

\[
F ::= \text{tt} \quad \text{(truth)}
\]

\[
| \quad \nabla \alpha \quad \text{(inability)}
\]

\[
| \quad \neg F \quad \text{(negation)}
\]

\[
| \quad F_1 \land F_2 \quad \text{(conjunction)}
\]

\[
| \quad \langle \alpha \rangle \mu F \quad \text{ (“at least”)}
\]
Relation to PEPA

**Defn.** \( P \xrightarrow{(\alpha, \nu)} S \) if for all \( P' \in S \), \( P \xrightarrow{\alpha} P' \) and
\[\sum\{ r \mid P \xrightarrow{(\alpha, r)} P', P' \in S \} = \nu.\]
Relation to PEPA

**Defn.** $P \xrightarrow{(\alpha,\nu)} S$ if for all $P' \in S$, $P \xrightarrow{\alpha} P'$ and

$$\sum\{r \mid P \xrightarrow{(\alpha,r)} P', P' \in S\} = \nu.$$

Let $P$ be a model of a PEPA process.

$$P \models \tt$$
Relation to PEPA

**Defn.** \( P \xrightarrow{(\alpha, \nu)} S \) if for all \( P' \in S \), \( P \xrightarrow{\alpha} P' \) and

\[
\sum \{ r \mid P \xrightarrow{(\alpha, r)} P', P' \in S \} = \nu.
\]

Let \( P \) be a model of a PEPA process.

\[
P \models \text{tt}
\]

\[
P \models \neg F \text{ if } P \not\models F
\]
Relation to PEPA

Defn. $P \xrightarrow{(\alpha, \nu)} S$ if for all $P' \in S$, $P \xrightarrow{\alpha} P'$ and
$$\sum\{ r \mid P \xrightarrow{(\alpha, r)} P', P' \in S \} = \nu.$$

Let $P$ be a model of a PEPA process.

- $P \models \mathsf{tt}$
- $P \models \neg F$ if $P \notmodels F$
- $P \models F_1 \land F_2$ if $P \models F_1$ and $P \models F_2$
Relation to PEPA

**Defn.** $P^{(\alpha,\nu)} \rightarrow S$ if for all $P' \in S$, $P^{(\alpha)} \rightarrow P'$ and 
$$\sum\{r \mid P^{(\alpha,r)} \rightarrow P', P' \in S\} = \nu.$$ 

Let $P$ be a model of a PEPA process.

- $P \models tt$
- $P \models \neg F$ if $P \not\models F$
- $P \models F_1 \land F_2$ if $P \models F_1$ and $P \models F_2$
- $P \models \nabla_{\alpha}$ if $P \xrightarrow{\alpha}$
### Relation to PEPA

**Defn.** \( P \xrightarrow{(\alpha, \nu)} S \) if for all \( P' \in S \), \( P \xrightarrow{\alpha} P' \) and
\[
\sum \{ r \mid P \xrightarrow{(\alpha, r)} P', P' \in S \} = \nu.
\]

Let \( P \) be a model of a PEPA process.

\[
P \models \top
\]

\[
P \models \neg F \text{ if } P \not\models F
\]

\[
P \models F_1 \land F_2 \text{ if } P \models F_1 \text{ and } P \models F_2
\]

\[
P \models \nabla \alpha \text{ if } P \xrightarrow{\alpha}
\]

\[
P \models \langle \alpha \rangle_{\mu} F \text{ if } P \xrightarrow{(\alpha, \nu)} S \text{ for some } \nu \geq \mu, \text{ and for all } P' \in S, P' \models F
\]
Let $P$ be a model of a PEPA process. Then

$$P \equiv Q \text{ iff for all } F, P \models F \text{ iff } Q \models F$$

That is to say that two PEPA processes are strongly equivalent (in particular, their underlying Markov chains are lumpably equivalent) if and only if they both satisfy, in the setting where rates are quantified, the same set of PML formulae.
We have used the PEPA modelling language and its accompanying specification language to analyse the configuration of a location tracking system based on active badges. Active badges transmit unique infra-red signals which are detected by networked sensors. These report locations back to a central database.
Case study: active badges

The badges are battery-powered and the tradeoff in the system is between the conservation of \textit{battery power} and the \textit{accuracy} of the information harvested from the sensors. When transmissions from badges collide, the badges sleep for a \textit{randomly determined} time before retrying.
The PEPA model of this system tracks the progress of one badge-wearer around three connected corridors (numbered 14, 15 and 16). The activities which are performed in the system include the badge registering with a sensor (at rate $r$), the person moving to another corridor (at rate $m$) and a sensor reporting back to the central database (at rate $s$).
Active badges: the PEPA model

Person

\[ P_{14} \overset{\text{def}}{=} (\text{reg}_{14}, r).P_{14} + (\text{move}_{15}, m).P_{15} \]
\[ P_{15} \overset{\text{def}}{=} (\text{reg}_{15}, r).P_{15} + (\text{move}_{14}, m).P_{14} + (\text{move}_{16}, m).P_{16} \]
\[ P_{16} \overset{\text{def}}{=} (\text{reg}_{16}, r).P_{16} + (\text{move}_{15}, m).P_{15} \]
Active badges: the PEPA model

**Person**

\[
\begin{align*}
P_{14} & \overset{\text{def}}{=} (\text{reg}_{14}, r).P_{14} + (\text{move}_{15}, m).P_{15} \\
P_{15} & \overset{\text{def}}{=} (\text{reg}_{15}, r).P_{15} + (\text{move}_{14}, m).P_{14} + (\text{move}_{16}, m).P_{16} \\
P_{16} & \overset{\text{def}}{=} (\text{reg}_{16}, r).P_{16} + (\text{move}_{15}, m).P_{15}
\end{align*}
\]

**Sensor**

\[
\begin{align*}
S_{14} & \overset{\text{def}}{=} (\text{reg}_{14}, \top).(\text{rep}_{14}, s).S_{14} \\
S_{15} & \overset{\text{def}}{=} (\text{reg}_{15}, \top).(\text{rep}_{15}, s).S_{15} \\
S_{16} & \overset{\text{def}}{=} (\text{reg}_{16}, \top).(\text{rep}_{16}, s).S_{16}
\end{align*}
\]
Active badges: the PEPA model

**Database**

\[
\begin{align*}
DB_{14} & \overset{\text{def}}{=} (rep_{14}, \top).DB_{14} + (rep_{15}, \top).DB_{15} + (rep_{16}, \top).DB_{16} \\
DB_{15} & \overset{\text{def}}{=} (rep_{14}, \top).DB_{14} + (rep_{15}, \top).DB_{15} + (rep_{16}, \top).DB_{16} \\
DB_{16} & \overset{\text{def}}{=} (rep_{14}, \top).DB_{14} + (rep_{15}, \top).DB_{15} + (rep_{16}, \top).DB_{16}
\end{align*}
\]
Active badges: the PEPA model

**Database**

\[
DB_{14} \overset{\text{def}}{=} (\text{rep}_{14}, \top).DB_{14} + (\text{rep}_{15}, \top).DB_{15} + (\text{rep}_{16}, \top).DB_{16}
\]

\[
DB_{15} \overset{\text{def}}{=} (\text{rep}_{14}, \top).DB_{14} + (\text{rep}_{15}, \top).DB_{15} + (\text{rep}_{16}, \top).DB_{16}
\]

\[
DB_{16} \overset{\text{def}}{=} (\text{rep}_{14}, \top).DB_{14} + (\text{rep}_{15}, \top).DB_{15} + (\text{rep}_{16}, \top).DB_{16}
\]

**System**

\[
P_{14} \bowtie_L (S_{14} \parallel S_{15} \parallel S_{16}) \bowtie_M DB_{14}
\]

where

\[
L = \{ \text{reg}_{14}, \text{reg}_{15}, \text{reg}_{16} \}
\]

\[
M = \{ \text{rep}_{14}, \text{rep}_{15}, \text{rep}_{16} \}
\]
Probability that the database holds inaccurate information

Stephen Gilmore. LFCS, University of Edinburgh.
Modelling Methods and Tools
Outline

1. A modelling language
2. A semantics for the modelling language
3. Tools for the modelling language
PEPA is a high-level language for performance modelling. PEPA models describe stochastic (in fact, Markovian) processes.
PEPA is a high-level language for performance modelling. PEPA models describe stochastic (in fact, Markovian) processes.

Without a high-level modelling language, the modeller would be forced to work with unstructured matrix representations of stochastic processes.
PEPA is a high-level language for performance modelling. PEPA models describe stochastic (in fact, Markovian) processes.

Without a high-level modelling language, the modeller would be forced to work with unstructured matrix representations of stochastic processes.

Process algebras are useful because they allow the definition of equivalence relations between model components and these relations may be used in model simplification.
The PEPA Workbench

Calculating by hand the transitions of a PEPA model and subsequently expressing these in a form which was suitable for solution was a tedious task prone to errors. The PEPA workbench relieves the modeller of this work.
The workbench will report errors in the model function:

- deadlock,
- absorbing states,
- static synchronisation mismatch (cooperations which do not involve active participants).

The workbench also generates the transition graph of the model, computes the number of states, formulates the Markov process matrix $Q$ and communicates the matrix to a solver.
The PEPA Workbench: functionality

The workbench will report errors in the model function:

- deadlock,
- absorbing states,
- static synchronisation mismatch (cooperations which do not involve active participants).

The workbench also generates the transition graph of the model, computes the number of states, formulates the Markov process matrix $Q$ and communicates the matrix to a solver.

The workbench provides a simple pattern language for selecting states from the stationary distribution.
PEPA Workbench input

\[ P_1 \overset{\text{def}}{=} (\text{start}, r_1).P_2 \quad P_2 \overset{\text{def}}{=} (\text{run}, r_2).P_3 \quad P_3 \overset{\text{def}}{=} (\text{stop}, r_3).P_1 \]

\[ P_1 \parallel P_1 \]
A modelling language
A semantics for the modelling language
Tools for the modelling language

PEPA Workbench input

\[ P_1 \overset{\text{def}}{=} (\text{start}, r_1).P_2 \]
\[ P_2 \overset{\text{def}}{=} (\text{run}, r_2).P_3 \]
\[ P_3 \overset{\text{def}}{=} (\text{stop}, r_3).P_1 \]

\[ P_1 \parallel P_1 \]

PEPA Workbench output

\[
\begin{pmatrix}
-2r_1 & r_1 & r_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -r_1 - r_2 & 0 & r_2 & r_1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -r_1 - r_2 & 0 & r_1 & r_2 & 0 & 0 & 0 & 0 \\
r_3 & 0 & 0 & -r_1 - r_3 & 0 & 0 & 0 & r_1 & 0 & 0 \\
0 & 0 & 0 & 0 & -2r_2 & 0 & r_2 & r_2 & 0 & 0 \\
r_3 & 0 & 0 & 0 & 0 & -r_1 - r_3 & r_1 & 0 & 0 & 0 \\
0 & r_3 & 0 & 0 & 0 & 0 & -r_2 - r_3 & 0 & r_2 & 0 \\
0 & 0 & r_3 & 0 & 0 & 0 & 0 & -r_2 - r_3 & r_2 & 0 \\
0 & 0 & 0 & r_3 & 0 & r_3 & 0 & 0 & 0 & -2r_3
\end{pmatrix}
\]
We are checking
for $P_1$
in
either case
test: $P_1|*$
test: $*|P_1$

\[
\text{test} := \text{proc } \pi ) ( \pi[1] \quad // \text{from } P_1 \parallel P_1 \\
+ \pi[3] \quad // \text{from } P_1 \parallel P_2 \\
+ \pi[6] \quad // \text{from } P_1 \parallel P_3 \\
+ \pi[2] \quad // \text{from } P_2 \parallel P_1 \\
+ \pi[4] \quad // \text{from } P_3 \parallel P_1 
\]
end:
Aggregation
We chose to implement the workbench in Standard ML, an imperative language with a functional sublanguage; a strong type system; a sophisticated module system and a formal definition which gives the definitive semantics.
We chose to implement the workbench in Standard ML, an imperative language with a functional sublanguage; a strong type system; a sophisticated module system and a formal definition which gives the definitive semantics.

Efficient model solution is essential for supporting a thorough experimental programme but this must be tempered with considerations of modeller convenience. Model solution methods are provided for the Maple, Matlab and Mathematica systems in addition to a C solver.
The PEPA syntax can be represented simply as an ML datatype.

\[
\text{datatype Component} = \\
\text{PREFIX of (Activity} \ast \text{Rate)} \ast \text{Component} (* . *) \\
\text{CHOICE of Component} \ast \text{Component} (* + *) \\
\text{COOP of Component} \ast \text{Component} \ast \text{Activity list} (* △ *) \\
\text{HIDING of Component} \ast \text{Activity list} (* / *) \\
\text{VAR of Identifier} (* X *) \\
\text{DEF of Identifier} \ast \text{Component} \ast \text{Component} (* \text{def} *)
\]
The PEPA syntax can be represented simply as an ML datatype.

```
datatype Component =
    PREFIX of (Activity * Rate) * Component (* . *)
  | CHOICE of Component * Component (* + *)
  | COOP of Component * Component * Activity list (* △ *)
  | HIDING of Component * Activity list (* / *)
  | VAR of Identifier (* X *)
  | DEF of Identifier * Component * Component (* def *)
```
The PEPA syntax can be represented simply as an ML datatype.

```ml
datatype Component =
  PREFIX of (Activity * Rate) * Component (* . *)
  | CHOICE of Component * Component (* + *)
  | COOP of Component * Component * Activity list (* ⊞ *)
  | HIDING of Component * Activity list (* / *)
  | VAR of Identifier (* X *)
  | DEF of Identifier * Component * Component (* def *)
```

Stephen Gilmore. LFCS, University of Edinburgh.

Modelling Methods and Tools
The PEPA syntax can be represented simply as an ML datatype.

```ml
datatype Component =
  PREFIX of (Activity * Rate) * Component (* . *)
  | CHOICE of Component * Component (* + *)
  | COOP of Component * Component * Activity list (* ⊢ *)
  | HIDING of Component * Activity list (* / *)
  | VAR of Identifier (* X *)
  | DEF of Identifier * Component * Component (* def *)
```

Stephen Gilmore. LFCS, University of Edinburgh.
The PEPA syntax can be represented simply as an ML datatype.

```
datatype Component =
    PREFIX of (Activity * Rate) * Component (* . *)
  | CHOICE of Component * Component (* + *)
  | COOP of Component * Component * Activity list (* △ *)
  | HIDING of Component * Activity list (* / *)
  | VAR of Identifier (* X *)
  | DEF of Identifier * Component * Component (* def *)
```
The PEPA syntax can be represented simply as an ML datatype.

``` ml
datatype Component =
  PREFIX of (Activity * Rate) * Component (* . *)
| CHOICE of Component * Component (* + *)
| COOP of Component * Component * Activity list (* ⊁ *)
| HIDING of Component * Activity list (* / *)
| VAR of Identifier (* X *)
| DEF of Identifier * Component * Component (* def *)
```
The PEPA syntax can be represented simply as an ML datatype.

```plaintext
datatype Component =
  PREFIX of (Activity * Rate) * Component
  | CHOICE of Component * Component
  | COOP of Component * Component * Activity list
  | HIDING of Component * Activity list
  | VAR of Identifier
  | DEF of Identifier * Component * Component
```

Stephen Gilmore. LFCS, University of Edinburgh.
The PEPA syntax can be represented simply as an ML datatype.

```plaintext
datatype Component =
      PREFIX of (Activity * Rate) * Component
    | CHOICE of Component * Component
    | COOP of Component * Component * Activity list
    | HIDING of Component * Activity list
    | VAR of Identifier
    | DEF of Identifier * Component * Component
```

Stephen Gilmore. LFCS, University of Edinburgh.
fun derivative E (PREFIX (a as (alpha, rate), P)) = [(a, P)]
| derivative E (CHOICE (P, Q)) =
   (derivative E P) @ (derivative E Q)
| derivative E (COOP (P, Q, L)) =
   let
   val (dP, dQ) = (derivative E P, derivative E Q)
   val (fP, fQ) = (filterout dP L, filterout dQ L)
   in
   (map (fn (a, P’) => (a, COOP (P’, Q, L))) fP)
   @ (map (fn (a, Q’) => (a, COOP (P, Q’, L))) fQ)
   @ cooperations dP dQ L
   end
| derivative E (HIDING (P, L)) = ...
fun derivative E (PREFIX (a as (alpha, rate), P)) = [(a, P)]
| derivative E (CHOICE (P, Q)) =
  (derivative E P) @ (derivative E Q)
| derivative E (COOP (P, Q, L)) =
  let
    val (dP, dQ) = (derivative E P, derivative E Q)
    val (fP, fQ) = (filterout dP L, filterout dQ L)
  in
    (map (fn (a, P’) => (a, COOP (P’, Q, L))) fP)
  @ (map (fn (a, Q’) => (a, COOP (P, Q’, L))) fQ)
  @ cooperations dP dQ L
  end
| derivative E (HIDING (P, L)) = ...
fun derivative E (PREFIX (a as (alpha, rate), P)) = [(a, P)]
  | derivative E (CHOICE (P, Q)) =
    (derivative E P) @ (derivative E Q)
  | derivative E (COOP (P, Q, L)) =
    let
      val (dP, dQ) = (derivative E P, derivative E Q)
      val (fP, fQ) = (filterout dP L, filterout dQ L)
    in
      (map (fn (a, P’) => (a, COOP (P’, Q, L))) fP)
    @ (map (fn (a, Q’) => (a, COOP (P, Q’, L))) fQ)
    @ cooperations dP dQ L
    end
  | derivative E (HIDING (P, L)) = ...

Stephen Gilmore. LFCS, University of Edinburgh. Modelling Methods and Tools
fun derivative E (PREFIX (a as (alpha, rate), P)) = [(a, P)]
| derivative E (CHOICE (P, Q)) =
   (derivative E P) @ (derivative E Q)
| derivative E (COOP (P, Q, L)) =
   let
     val (dP, dQ) = (derivative E P, derivative E Q)
     val (fP, fQ) = (filterout dP L, filterout dQ L)
   in
     (map (fn (a, P’) => (a, COOP (P’, Q, L))) fP)
     @ (map (fn (a, Q’) => (a, COOP (P, Q’, L))) fQ)
     @ cooperations dP dQ L
   end
| derivative E (HIDING (P, L)) = ...
fun derivative E (PREFIX (a as (alpha, rate), P)) = [(a, P)]
| derivative E (CHOICE (P, Q)) =
    (derivative E P) @ (derivative E Q)
| derivative E (COOP (P, Q, L)) =
    let
        val (dP, dQ) = (derivative E P, derivative E Q)
        val (fP, fQ) = (filterout dP L, filterout dQ L)
    in
        (map (fn (a, P’) => (a, COOP (P’, Q, L))) fP)
    @ (map (fn (a, Q’) => (a, COOP (P, Q’, L))) fQ)
    @ cooperations dP dQ L
    end
| derivative E (HIDING (P, L)) = ...
With the ML edition of the PEPA Workbench it was possible to solve small models using exterior solvers such as Maple and Matlab.

However, users of the workbench wanted to make more detailed models (with larger state spaces).

The ML edition of the PEPA Workbench could not solve Robert Holton’s robotic workcell model efficiently enough so we interfaced it with an external solver written in C.

Other users wanted to run the workbench on Solaris, Windows and Linux machines so we ported the Workbench and the solver to Java.