Performance Modelling of Computer Systems

Mirco Tribastone

Institut für Informatik
Ludwig-Maximilians-Universität München

High-Level Modelling Techniques

Stochastic Process Algebra

Overview

- Overview of classic (untimed) process algebra
- Associating exponential distributions to activities
- Introduction to the stochastic process algebra PEPA

Bibliographic references:

- A. Clark, J. Hillston, and M. Tribastone. **Stochastic Process Algebras**. In *Formal Methods for Performance Evaluation: the 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007*, LNCS 4486, Springer-Verlag.
- J. Hillston. **A Compositional Approach to Performance Modelling**. Cambridge University Press, 1996.

Motivation

- High-level formalisms for performance evaluation (such as queueing networks and stochastic Petri nets) satisfy the need to abstract away from the stochastic process which represents the system under scrutiny.
- Process algebras offer the additional advantage of being compositional.
- The system is constructed in a modular way by composing communicating components.
- The reasoning is also modular. From properties of the individual components one can infer properties that hold in the system.

Classic Process Algebras

- The Calculus of Communicating Systems by Robin Milner and Communicating Sequential Processes by Tony Hoare are the pioneering works in the context of classic process algebras.
- They were developed in the 80's for qualitative reasoning about the behaviour of distributed computation.

Formal Definition of CCS

Syntax of CCS

Prefix	
$a.B$	after action a the agent becomes B
Constant	
$K \stackrel{def}{=} P$	assigns the name K to agent P
Parallel composition	
$A \mid B$	agents A and B proceed in parallel
Choice	
$A + B$	the agent behaves as A or B depending on which acts first
Restriction	
$A \backslash M$	the set of labels M is hidden from outside agents
Relabelling	
$A[a_1/a_0, ..]$	in this agent label a_1 is renamed a_0
Null agent	
$\mathbf{0}$	this agent cannot act (deadlock)

Example

Recalling the consumer-producer problem examined in the previous tutorial, one may model the system components as follows:

$$
\begin{aligned}
\mathit{Producer} &\stackrel{\mathit{def}}{=} \mathit{canProduce}.\mathit{doProduce}.\mathit{Producer} \\
\mathit{Consumer} &\stackrel{\mathit{def}}{=} \mathit{canConsume}.\mathit{doConsume}.\mathit{Consumer} \\
\mathit{Buffer}_2 &\stackrel{\mathit{def}}{=} \overline{\mathit{canConsume}}.\mathit{Buffer}_1 \\
\mathit{Buffer}_1 &\stackrel{\mathit{def}}{=} \overline{\mathit{canProduce}}.\mathit{Buffer}_2 + \overline{\mathit{canConsume}}.\mathit{Buffer}_0 \\
\mathit{Buffer}_0 &\stackrel{\mathit{def}}{=} \overline{\mathit{canProduce}}.\mathit{Buffer}_1 \\
\mathit{System} &\stackrel{\mathit{def}}{=} \Big((\mathit{Producer} \mid \mathit{Buffer}_2) \\
&\qquad\qquad \mid \mathit{Consumer} \Big) \setminus \{\mathit{canConsume}, \mathit{canProduce}\}
\end{aligned}
$$

Producer and *Consumer* are usually called sequential agents whereas *System* is called a compound agent or process

Semantics of Process Algebra

An operational semantics allows the interpretation of process algebra model as a labelled transition system (LTS).
For example we will be able to write transitions of kind

$$\Big((Producer|Buffer_2)|Consumer\Big) \xrightarrow{\tau}$$
$$\Big((Producer|Buffer_1)|doConsume.Consumer\Big) \xrightarrow{doConsume}$$
$$\Big((Producer|Buffer_1)|Consumer\Big) \xrightarrow{\cdots} \ldots$$

SOS

Process algebras often use a structured operational semantics, a collection of inference rules of kind

$$\frac{\text{premise}}{\text{conclusion}}, \qquad \text{that is, } \textbf{if} \text{ premise } \textbf{then} \text{ conclusion.}$$

(An inference rule with no premise will be an axiom of the language.)

Structured Operational Semantics

Let $\mathcal{A}$ be the set of action names and $\overline{\mathcal{A}}$ be the set of co-names, ranged over by a and $\overline{a}$, respectively. Let $\mathrm{Act} = \mathcal{A} \cup \overline{\mathcal{A}} \cup \{\tau\}$, ranged over by α. (We also assume that $\overline{\overline{a}} = a$.)

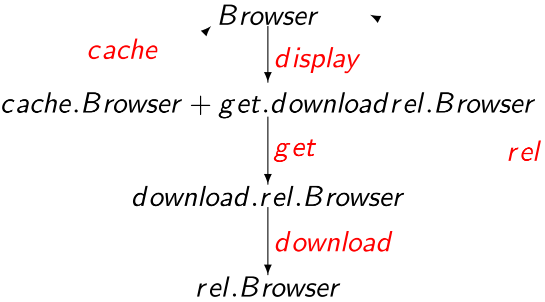
Semantics for a sequential agent:

$$Browser \stackrel{def}{=} display.(cache.Browser \;+\; get.download.rel.Browser)$$

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$



Semantics of Communication

Concurrent Actions

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \text{and} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}$$

Synchronised Actions

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

Hiding

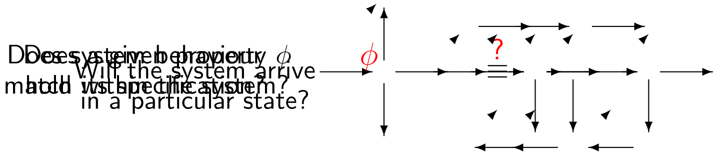
$$\frac{P \xrightarrow{\alpha} P'}{P \backslash M \xrightarrow{\overline{\alpha}} P' \backslash M}, \qquad \alpha, \overline{\alpha} \notin M$$

Example

Let us compute the LTS for the producer/consumer model:

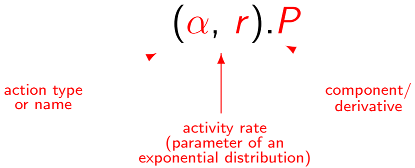
$$
\begin{aligned}
\mathit{Producer} &\stackrel{\text{def}}{=} \mathit{canProduce}.\mathit{doProduce}.\mathit{Producer} \\
\mathit{Consumer} &\stackrel{\text{def}}{=} \mathit{canConsume}.\mathit{doConsume}.\mathit{Consumer} \\
\mathit{Buffer}_2 &\stackrel{\text{def}}{=} \overline{\mathit{canConsume}}.\mathit{Buffer}_1 \\
\mathit{Buffer}_1 &\stackrel{\text{def}}{=} \overline{\mathit{canProduce}}.\mathit{Buffer}_2 + \overline{\mathit{canConsume}}.\mathit{Buffer}_0 \\
\mathit{Buffer}_0 &\stackrel{\text{def}}{=} \overline{\mathit{canProduce}}.\mathit{Buffer}_1 \\
\mathit{System} &\stackrel{\text{def}}{=} \Big(\big(\mathit{Producer} \mid \mathit{Buffer}_2\big) \\
&\qquad\qquad \mid \mathit{Consumer}\Big) \setminus \{\mathit{canConsume}, \mathit{canProduce}\}
\end{aligned}
$$

- The labelled transition system underlying a process algebra model can be used for functional verification e.g.: reachability analysis, specification matching and model checking.



Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.



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

PEPA

BNF Syntax

$$
\begin{array}{lcl}
S & ::= & (\alpha, r).S \mid S + S \mid A \\
P & ::= & S \mid P \underset{L}{\bowtie} P \mid P/L
\end{array}
$$

PREFIX:	$(\alpha, r).S$	designated first action
CHOICE:	$S + S$	competing components (race policy)
CONSTANT:	$A \stackrel{def}{=} S$	assigning names
COOPERATION:	$P \underset{L}{\bowtie} P$	$\alpha \notin L$ concurrent activity (*individual actions*) $\alpha \in L$ cooperative activity (*shared actions*)
HIDING:	P/L	abstraction $\alpha \in L \Rightarrow \alpha \rightarrow \tau$

Comments

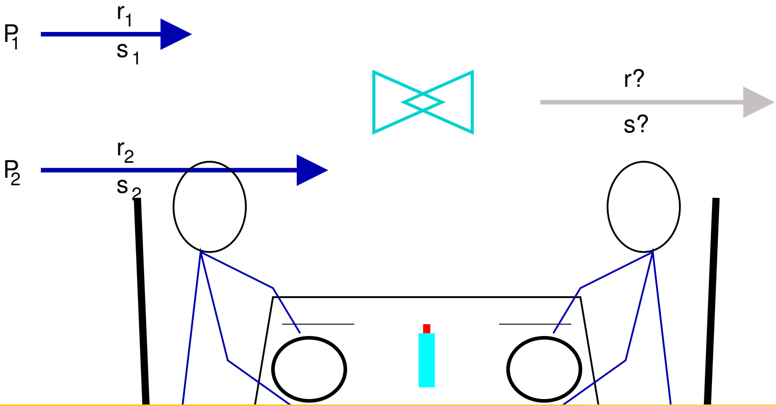
- PEPA is derived from the process algebra CSP.
- Although some elements are common to CCS, the semantics of synchronisation differs substantially.
- In PEPA, the synchronisation set is explicit.
- There are no co-names, but synchronisation occurs over shared actions. Unlike CSP, which produces a silent action as the result of synchronisation, in PEPA the action type is preserved.
- For instance, the following transition can be proven:

$$(\alpha, r_1).P_1 \underset{\{\alpha\}}{\bowtie} (\alpha, r_2).P_2 \xrightarrow{(\alpha, r)} P_1 \underset{\{\alpha\}}{\bowtie} P_2$$

- In addition to the action type, the transition is labelled with a resulting rate of execution.

Timed Synchronisation

- The issue of what it means for two timed activities to synchronise is a vexed one...



Cooperation in PEPA

- In PEPA each component has a bounded capacity to carry out activities of any particular type, determined by the apparent rate for that type.
- Synchronisation, or cooperation cannot make a component exceed its bounded capacity.
- Thus the apparent rate of a cooperation is the minimum of the apparent rates of the co-operands.

Operational Semantics of PEPA

$$\mathrm{S}_0: \quad \frac{}{(\alpha,r).P \xrightarrow{(\alpha,r)} P} \qquad\qquad \mathrm{A}_0: \quad \frac{P \xrightarrow{(\alpha,r)} P'}{A \xrightarrow{(\alpha,r)} P'},\ A \stackrel{\mathit{def}}{=} P$$

$$\mathrm{S}_1: \quad \frac{P \xrightarrow{(\alpha,r)} P'}{P+Q \xrightarrow{(\alpha,r)} P'+Q} \qquad\qquad \mathrm{S}_2: \quad \frac{Q \xrightarrow{(\alpha,r)} Q'}{P+Q \xrightarrow{(\alpha,r)} P+Q'}$$

$$\mathrm{C}_0: \quad \frac{P \xrightarrow{(\alpha,r)} P'}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r)} P' \underset{L}{\bowtie} Q},\ \alpha \notin L \qquad\qquad \mathrm{C}_1: \quad \frac{Q \xrightarrow{(\alpha,r)} Q'}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r)} P \underset{L}{\bowtie} Q'},\ \alpha \notin L$$

$$\mathrm{C}_2: \quad \frac{P \xrightarrow{(\alpha,r_1)} P' \qquad Q \xrightarrow{(\alpha,r_2)} Q'}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,R)} P' \underset{L}{\bowtie} Q'},\ \alpha \in L \qquad\qquad R = \frac{r_1}{r_\alpha(P)} \frac{r_2}{r_\alpha(Q)} \min\left(r_\alpha(P), r_\alpha(Q)\right)$$

$$\mathrm{H}_0: \quad \frac{P \xrightarrow{(\alpha,r)} P'}{P/L \xrightarrow{(\alpha,r)} P'/L},\ \alpha \notin L \qquad\qquad \mathrm{H}_1: \quad \frac{P \xrightarrow{(\alpha,r)} P'}{P/L \xrightarrow{(\tau,r)} P'/L},\ \alpha \in L$$

Multiway Synchronisation

$$Fork \stackrel{def}{=} (fork, r_f).(join, r_j)\ldots$$
$$W_1 \stackrel{def}{=} (fork, r_{f_1}).(doWork_1, r_1)\ldots$$
$$W_2 \stackrel{def}{=} (fork, r_{f_2}).(doWork_2, r_2)\ldots$$
$$System \stackrel{def}{=} \left(Fork \underset{\{fork\}}{\bowtie} W_1\right) \underset{\{fork\}}{\bowtie} W_2$$

$$\frac{P \xrightarrow{(\alpha,r)} P'}{A \xrightarrow{(\alpha,r)} P'}, \; A \stackrel{def}{=} P \Longrightarrow$$

1. $$\frac{(fork, r_f).(join, r_j)\ldots \xrightarrow{(fork,r_f)} (join, r_j)\ldots}{Fork \xrightarrow{(fork,r_f)} (join, r_j)\ldots}$$

2. $$\frac{(fork, r_{f_1}).(doWork_1, r_1)\ldots \xrightarrow{(fork,r_{f_1})} (doWork_1, r_1)\ldots}{W_1 \xrightarrow{(fork,r_{f_1})} (doWork_1, r_1)\ldots}$$

3. $$\frac{(fork, r_{f_2}).(doWork_2, r_2)\ldots \xrightarrow{(fork,r_{f_2})} (doWork_2, r_2)\ldots}{W_2 \xrightarrow{(fork,r_{f_2})} (doWork_2, r_2)\ldots}$$

Multiway Synchronisation

$$
\begin{aligned}
Fork &\stackrel{def}{=} (fork, r_f).(join, r_j)\ldots \\
W_1 &\stackrel{def}{=} (fork, r_{f_1}).(doWork_1, r_1)\ldots \\
W_2 &\stackrel{def}{=} (fork, r_{f_2}).(doWork_2, r_2)\ldots \\
System &\stackrel{def}{=} \left(Fork \underset{\{fork\}}{\bowtie} W_1\right) \underset{\{fork\}}{\bowtie} W_2
\end{aligned}
$$

$$
\frac{Fork \xrightarrow{(fork, r_f)} (join, r_j)\ldots \qquad W_1 \xrightarrow{(fork, r_{f_1})} (doWork_1, r_1)\ldots}{Fork \underset{\{fork\}}{\bowtie} W_1 \xrightarrow{(fork, r')} (join, r_j)\ldots \underset{\{fork\}}{\bowtie} (doWork_1, r_1)\ldots \equiv LHS}
$$

$$
\frac{LHS \qquad W_2 \xrightarrow{(fork, r_{f_2})} (doWork_2, r_2)\ldots}{Fork \underset{\{fork\}}{\bowtie} W_1 \underset{\{fork\}}{\bowtie} W_2 \xrightarrow{(fork, r'')} (join, r_j)\ldots \underset{\{fork\}}{\bowtie} (doWork_1, r_1)\ldots \underset{\{fork\}}{\bowtie} (doWork_2, r_2)\ldots}
$$

Performance Evaluation
Process Algebra

Other Communication Patterns

$$Premium \stackrel{def}{=} (dwn, r_p).Premium'$$
$$Basic \stackrel{def}{=} (dwn, r_b).Basic'$$
$$S \stackrel{def}{=} (dwn, r_s).S'$$
$$\ldots$$
$$System \stackrel{def}{=} (Premium \parallel Basic) \underset{L}{\bowtie} S,$$
$$L = \{dwn\}$$

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r)} P' \underset{L}{\bowtie} Q}, \alpha \notin L$$

$$\frac{Q \xrightarrow{(\alpha,r)} Q'}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r)} P \underset{L}{\bowtie} Q'}, \alpha \notin L$$

$$\frac{P \xrightarrow{(\alpha,r_1)} P' \quad Q \xrightarrow{(\alpha,r_2)} Q'}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,R)} P' \underset{L}{\bowtie} Q'}, \alpha \in L$$

$$R = \frac{r_1}{r_\alpha(P)} \frac{r_2}{r_\alpha(Q)} \min\left(r_\alpha(P), r_\alpha(Q)\right)$$

$$\cfrac{\cfrac{\cfrac{Premium \xrightarrow{(dwn,r_p)} Premium'}{Premium \parallel Basic \xrightarrow{(dwn,r_p)} Premium' \parallel Basic} \qquad \cfrac{}{S \xrightarrow{(dwn,r_s)} S'}}{Premium \parallel Basic \underset{L}{\bowtie} S \xrightarrow{(dwn,\color{red}{r_{ps}})} Premium' \parallel Basic \underset{L}{\bowtie} S'}}{\color{red}{System \xrightarrow{(dwn,r_{ps})} Premium' \parallel Basic \underset{L}{\bowtie} S'}}$$

$$Basic \xrightarrow{(dwn,r_b)} Basic'$$

Rates in PEPA

PEPA supports the notion of infinite capacity:

$$(\alpha, r).P, \qquad \text{with } r \in \mathbb{R}_{>0} \cup \{n\top, n \in \mathbb{N}\}.$$

- A positive real denotes the rate of the exponential distribution associated with the activity.
- The top symbol $\top$ denotes an unspecified (or passive) rate. The rate will be assigned by other cooperating components in the system.
- Passive rates are given weights (naturals) which are useful to determine the relative probabilities of distinct passive activities to occur. ($1\top$ is usually written $\top$ for short.)

Arithmetic for Passive Rates

$$\begin{aligned}
m\top + n\top &= (m+n)\top, && \text{for any } m, n \in \mathbb{N} \\
\frac{m\top}{n\top} &= \frac{m}{n}, && \text{for any } m, n \in \mathbb{N} \\
\min(r, n\top) &= r, && \text{for any } r \in \mathbb{R}_{>0} \text{ and } n \in \mathbb{N} \\
\min(m\top, n\top) &= \min(m, n)\top, && \text{for any } m, n \in \mathbb{N}
\end{aligned}$$

- Summation and division between active and passive rates are not allowed.
- For expression of the following kind:

$$\frac{r}{s} \times \frac{m\top}{n\top}, \qquad r, s \in \mathbb{R}_{>0}, m, n \in \mathbb{N}$$

we assume that the two divisions have precedence over the multiplication.

Apparent Rate Calculation

$$\frac{P \xrightarrow{(\alpha,r_1)} P' \quad Q \xrightarrow{(\alpha,r_2)} Q'}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,R)} P' \underset{L}{\bowtie} Q'}, \alpha \in L, \qquad R = \frac{r_1}{r_\alpha(P)} \frac{r_2}{r_\alpha(Q)} \min\left(r_\alpha(P), r_\alpha(Q)\right)$$

$$r_\alpha\left((\beta, r).P \right) = \begin{cases} r & \text{if} \quad \beta = \alpha \\ 0 & \text{if} \quad \beta \neq \alpha \end{cases}$$

$$r_\alpha(P + Q) = r_\alpha(P) + r_\alpha(Q)$$

$$r_\alpha(P \underset{L}{\bowtie} Q) = \begin{cases} \min\left(r_\alpha(P), r_\alpha(Q)\right) & \text{if } \alpha \in L \\ r_\alpha(P) + r_\alpha(Q) & \text{if } \alpha \notin L \end{cases}$$

$$r_\alpha(P/L) = \begin{cases} r_\alpha(P) & \text{if} \quad \alpha \notin L \\ 0 & \text{if} \quad \alpha \in L \end{cases}$$

Components which are both active and passive with respect to some action type are not allowed, e.g. $(\alpha, 1.0).P + (\alpha, \top).P$.

Examples

For r_1, r_2 positive reals,

$$\frac{(\alpha, r_1).P_1 \xrightarrow{(\alpha,r_1)} P_1 \qquad (\alpha, r_2).P_2 \xrightarrow{(\alpha,r_2)} P_2}{(\alpha, r_1).P_1 \underset{\{\alpha\}}{\bowtie} (\alpha, r_2).P_2 \xrightarrow{(\alpha,R)} P_1 \underset{\{\alpha\}}{\bowtie} P_2},$$

where

$$\begin{aligned} R &= \frac{r_1}{r_\alpha\big((\alpha, r_1).P_1\big)} \frac{r_2}{r_\alpha\big((\alpha, r_2).P_2\big)} \min\Big(r_\alpha\big((\alpha, r_1).P_1\big), r_\alpha\big((\alpha, r_2).P_2\big)\Big) \\ &= \frac{r_1}{r_1}\frac{r_2}{r_2}\min(r_1, r_2) = \min(r_1, r_2). \end{aligned}$$

We recover the intuitive definition of the minimum between the two rates.

Examples

For r a positive real,

$$\frac{(\alpha, r).P_1 \xrightarrow{(\alpha,r)} P_1 \qquad (\alpha, \top).P_2 \xrightarrow{(\alpha,\top)} P_2}{(\alpha, r).P_1 \underset{\{\alpha\}}{\bowtie} (\alpha, \top).P_2 \xrightarrow{(\alpha,R)} P_1 \underset{\{\alpha\}}{\bowtie} P_2},$$

where

$$\begin{aligned} R &= \frac{r}{r_\alpha\big((\alpha, r).P_1\big)} \frac{\top}{r_\alpha\big((\alpha, \top).P_2\big)} \min\Big(r_\alpha\big((\alpha, r).P_1\big), r_\alpha\big((\alpha, \top).P_2\big)\Big) \\ &= \frac{r}{r}\frac{\top}{\top}\min(r, \top) = r. \end{aligned}$$

We recover the intuitive definition of infinite capacity — the rate of synchronisation is determined by the active component.

Examples

For r a positive real and any natural n,

$$\frac{(\alpha, r).P_1 \xrightarrow{(\alpha,r)} P_1 \qquad (\alpha, n\top).P_2 \xrightarrow{(\alpha,n\top)} P_2}{(\alpha, r).P_1 \underset{\{\alpha\}}{\bowtie} (\alpha, n\top).P_2 \xrightarrow{(\alpha,R)} P_1 \underset{\{\alpha\}}{\bowtie} P_2},$$

where

$$\begin{aligned} R &= \frac{r}{r_\alpha\big((\alpha, r).P_1\big)} \frac{n\top}{r_\alpha\big((\alpha, n\top).P_2\big)} \min\Big(r_\alpha\big((\alpha, r).P_1\big), r_\alpha\big((\alpha, n\top).P_2\big)\Big) \\ &= \frac{r}{r}\frac{n\top}{n\top}\min(r, n\top) = r. \end{aligned}$$

Passive weights may not affect the overall rate if only one passive component is present.

(Slightly More Complicated) Examples

$$Act \stackrel{def}{=} (\alpha, r).Act'$$
$$Pas \stackrel{def}{=} (\alpha, 1\top).Pas' + (\alpha, 2\top).Pas''$$
$$Sys \stackrel{def}{=} Act \underset{\{\alpha\}}{\bowtie} Pas$$

$$\frac{\dfrac{(\alpha, r).Act' \xrightarrow{(\alpha,r)} Act'}{Act \xrightarrow{(\alpha,r)} Act'} \qquad \dfrac{\dfrac{\overline{(\alpha, 1\top).Pas' \xrightarrow{(\alpha,1\top)} Pas'}}{(\alpha, 1\top).Pas' + (\alpha, 2\top).Pas'' \xrightarrow{(\alpha,1\top)} Pas'}}{Pas \xrightarrow{(\alpha,1\top)} Pas'}}{\dfrac{Act \underset{\{\alpha\}}{\bowtie} Pas \xrightarrow{(\alpha,R')} Act' \underset{\{\alpha\}}{\bowtie} Pas'}{Sys \xrightarrow{(\alpha,R')} Act' \underset{\{\alpha\}}{\bowtie} Pas'}},$$

$$R' = \frac{r}{r_\alpha(Act)} \frac{1\top}{r_\alpha(Pas)} \min\left(r_\alpha(Act), r_\alpha(Pas)\right) = \frac{r}{r} \frac{1\top}{1\top + 2\top} \min(r, 1\top+2\top) = \frac{1}{3} r.$$

(Slightly More Complicated) Examples

$$
\begin{aligned}
Act &\stackrel{def}{=} (\alpha, r).Act' \\
Pas &\stackrel{def}{=} (\alpha, 1\top).Pas' + (\alpha, 2\top).Pas'' \\
Sys &\stackrel{def}{=} Act \underset{\{\alpha\}}{\bowtie} Pas
\end{aligned}
$$

It is also possible to prove the following derivation tree:

$$
\dfrac{\dfrac{\dfrac{}{(\alpha, r).Act' \xrightarrow{(\alpha,r)} Act'}}{Act \xrightarrow{(\alpha,r)} Act'} \qquad \dfrac{\dfrac{\dfrac{}{(\alpha, 2\top).Pas'' \xrightarrow{(\alpha,2\top)} Pas''}}{(\alpha, 1\top).Pas' + (\alpha, 2\top).Pas'' \xrightarrow{(\alpha,2\top)} Pas''}}{Pas \xrightarrow{(\alpha,2\top)} Pas''}}{\dfrac{Act \underset{\{\alpha\}}{\bowtie} Pas \xrightarrow{(\alpha,R'')} Act' \underset{\{\alpha\}}{\bowtie} Pas''}{Sys \xrightarrow{(\alpha,R'')} Act' \underset{\{\alpha\}}{\bowtie} Pas''}},
$$

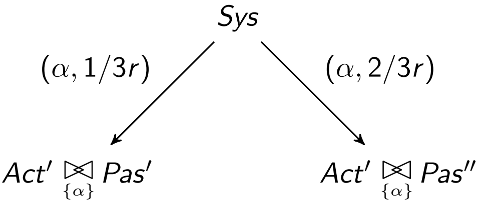
$$
R'' = \frac{r}{r_\alpha(Act)} \frac{2\top}{r_\alpha(Pas)} \min\left(r_\alpha(Act), r_\alpha(Pas)\right) = \frac{r}{r} \frac{2\top}{1\top + 2\top} \min(r, 1\top+2\top) = \frac{2}{3} r.
$$

(Slightly More Complicated) Examples

$$Act \stackrel{def}{=} (\alpha, r).Act'$$

$$Pas \stackrel{def}{=} (\alpha, 1\top).Pas' + (\alpha, 2\top).Pas''$$

$$Sys \stackrel{def}{=} Act \underset{\{\alpha\}}{\bowtie} Pas$$



Apparent Rates in Active Cooperation

$$Cli \stackrel{def}{=} (\alpha, r_d).Cli'$$
$$Ser \stackrel{def}{=} (\alpha, r_u).Ser'$$
$$Sys \stackrel{def}{=} (Cli \parallel Cli) \underset{\{\alpha\}}{\bowtie} Ser$$

$$\frac{\dfrac{\dfrac{\overline{(\alpha, r_d).Cli' \xrightarrow{(\alpha, r_d)} Cli'}}{Cli \xrightarrow{(\alpha, r_d)} Cli'}}{Cli \parallel Cli \xrightarrow{(\alpha, r_d)} Cli' \parallel Cli} \qquad \dfrac{(\alpha, r_u).Ser' \xrightarrow{(\alpha, r_u)} Ser'}{Ser \xrightarrow{(\alpha, r_u)} Ser'}}{Cli \parallel Cli \underset{\{\alpha\}}{\bowtie} Ser \xrightarrow{(\alpha, R')} Cli' \parallel Cli \underset{\{\alpha\}}{\bowtie} Ser'},$$

$$R' = \frac{r_d}{r_d + r_d} \frac{r_u}{r_u} \min(r_d + r_d, r_u) = \frac{1}{2} \min(r_d + r_d, r_u)$$

Apparent Rates in Active Cooperation

$$
\begin{array}{|l|}
\hline
Cli \stackrel{def}{=} (\alpha, r_d).Cli' \\
Ser \stackrel{def}{=} (\alpha, r_u).Ser' \\
Sys \stackrel{def}{=} (Cli \parallel Cli) \underset{\{\alpha\}}{\bowtie} Ser \\
\hline
\end{array}
$$

The following derivation tree can also be proven:

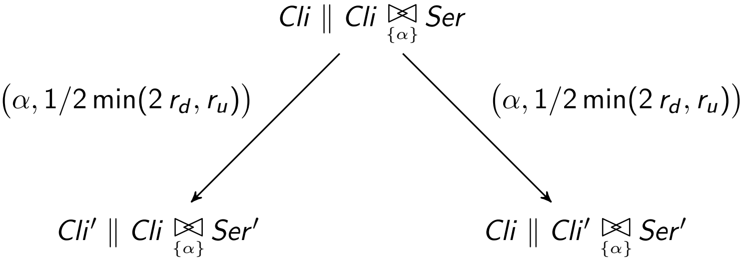
$$
\frac{
\dfrac{\dfrac{\dfrac{}{(\alpha, r_d).Cli' \xrightarrow{(\alpha, r_d)} Cli'}}{Cli \xrightarrow{(\alpha, r_d)} Cli'}}{\color{red}{Cli \parallel Cli \xrightarrow{(\alpha, r_d)} Cli \parallel Cli'}}
\qquad
\dfrac{(\alpha, r_u).Ser' \xrightarrow{(\alpha, r_u)} Ser'}{Ser \xrightarrow{(\alpha, r_u)} Ser'}
}{
Cli \parallel Cli \underset{\{\alpha\}}{\bowtie} Ser \xrightarrow{(\alpha, R'')} Cli \parallel Cli' \underset{\{\alpha\}}{\bowtie} Ser'
},
$$

$$
R'' = \frac{r_d}{r_d + r_d} \frac{r_u}{r_u} \min(r_d + r_d, r_u) = \frac{1}{2} \min(r_d + r_d, r_u) = R'
$$

$$Cli \stackrel{def}{=} (\alpha, r_d).Cli'$$

$$Ser \stackrel{def}{=} (\alpha, r_u).Ser'$$

$$Sys \stackrel{def}{=} (Cli \parallel Cli) \underset{\{\alpha\}}{\bowtie} Ser$$



Labelled Transition System: Details

Derivative Set

Given a PEPA component P, the derivative set of P, denoted by $ds(P)$ is defined as the smallest set of components such that

- $P \in ds(P)$;
- if $P \xrightarrow{(\alpha,r)} P'$ then $P' \in ds(P)$.

Derivation Graph

Let $\mathcal{A}$ be a set of action labels and $\mathcal{A}ct = \{\!| (\alpha, r) : \alpha \in \mathcal{A}, r \in \mathbb{R}_{>0} |\!\}$. The derivation graph of a component P has $ds(P)$ as the set of nodes.
The multiset of arcs $A \in ds(P) \times ds(P) \times \mathcal{A}ct$ is such that

$$P \xrightarrow{(\alpha,r)} P' \implies \big(P, P', (\alpha, r)\big) \in A,$$

with multiplicity equal to the number of distinct derivations $P \xrightarrow{(\alpha,r)} P'$.

Why Multisets

$$P \stackrel{\text{def}}{=} (\alpha, r).P' \;\Big|\; P \stackrel{\text{def}}{=} (\alpha, r).P' + (\alpha, r).P' \;\Big|\; \ldots \;\Big|\; P \stackrel{\text{def}}{=} \sum_n (\alpha, r).P'$$

- If distinct inference trees were not taken into account, then the derivation graph would have only one transition $P \xrightarrow{(\alpha,r)} P'$.
- With a multiset, we have one, two, ..., n such transitions, respectively.
- Intuitively, this capture the fact that process P has different apparent rates in these cases.

An Algorithm for State-Space Derivation

```
ds(P_0) ⇐ {P_0}
push P_0 onto Stack
while Stack is not empty do
   pop P off Stack
   infer multiset (P, P′, (α, r)) from P
   for all (P, P′, (α, r)) do
      if P′ ∉ ds(P_0) then
         push P′ onto Stack
         add P′ to ds(P_0)
      end if
   end for
end while
```

The Underlying Markov Process

- Let P_0 be the initial state of the system.
- Assign a state to each process in $\mathrm{ds}(P_0)$.
- For each triple $\big(P, P', (\alpha, r)\big)$ with multiplicity m, assign rate $m\,r$ to the transition between P and P'.

Well-Formedness

- Note that all leaves of the derivation trees must have rates in the (strictly) positive reals.
- This means that passive actions must eventually synchronise with an active ones.
- Models that do not satisfy this condition are rejected.
- For example,

$$(\alpha, \top).P \underset{\{\alpha\}}{\bowtie} (\alpha, \top).Q$$

will be rejected for any P and Q.

Consumer/Producer in PEPA

$$
\begin{aligned}
Cons_1 &\stackrel{def}{=} (get, r_g).Cons_2 \\
Cons_2 &\stackrel{def}{=} (cons, r_c).Cons_1 \\
Prod_1 &\stackrel{def}{=} (make, r_m).Prod_2 \\
Prod_2 &\stackrel{def}{=} (put, r_p).Prod_1 \\
Buf_2 &\stackrel{def}{=} (get, \top).Buf_1 \\
Buf_1 &\stackrel{def}{=} (get, \top).Buf_0 \\
&\qquad + (put, \top).Buf_2 \\
Buf_0 &\stackrel{def}{=} (put, \top).Buf_1 \\
Sys &\stackrel{def}{=} Cons_1 \underset{\{get\}}{\bowtie} Buf_2 \underset{\{put\}}{\bowtie} Prod_1
\end{aligned}
$$

Possible variants:

- A buffer with n places:

$$
\begin{aligned}
Buf_n &\stackrel{def}{=} (get, \top).Buf_{n-1} \\
Buf_i &\stackrel{def}{=} (get, \top).Buf_{i-1} \\
&\qquad + (put, \top).Buf_{i+1}, \\
&\qquad \text{for } 1 \leq i \leq n-1 \\
Buf_0 &\stackrel{def}{=} (put, \top).Buf_1
\end{aligned}
$$

- and k consumers:

$$
\overbrace{Cons_1 \parallel Cons_1 \parallel \ldots \parallel Cons_1}^{k} \underset{\{get\}}{\bowtie} Buf_n \underset{\{put\}}{\bowtie} Prod_1
$$

Consumer/Producer in PEPA

$$
\begin{array}{llllll}
Cons_1 & \stackrel{def}{=} & (get, r_g).Cons_2 & Prod_1 & \stackrel{def}{=} & (make, r_m).Prod_2 \\
Cons_2 & \stackrel{def}{=} & (cons, r_c).Cons_1 & Prod_2 & \stackrel{def}{=} & (put, r_p).Prod_1 \\
Buf_2 & \stackrel{def}{=} & (get, \top).Buf_1 & Buf_1 & \stackrel{def}{=} & (get, \top).Buf_0 + (put, \top).Buf_2 \\
Buf_0 & \stackrel{def}{=} & (put, \top).Buf_1 & Sys & \stackrel{def}{=} & Cons_1 \bowtie_{\{get\}} Buf_2 \bowtie_{\{put\}} Prod_1
\end{array}
$$

$$
\dfrac{\dfrac{\dfrac{Cons_1 \xrightarrow{(get,r_g)} Cons_2 \qquad Buf_2 \xrightarrow{(get,\top)} Buf_1}{Cons_1 \bowtie_{\{get\}} Buf_2 \xrightarrow{(get,r_g)} Cons_2 \bowtie_{\{get\}} Buf_1}}{Cons_1 \bowtie_{\{get\}} Buf_2 \bowtie_{\{put\}} Prod_1 \xrightarrow{(get,r_g)} Cons_2 \bowtie_{\{get\}} Buf_1 \bowtie_{\{put\}} Prod_1}}{Sys \xrightarrow{(get,r_g)} Cons_2 \bowtie_{\{get\}} Buf_1 \bowtie_{\{put\}} Prod_1}
$$

Can we prove anything else for Sys?

Consumer/Producer in PEPA

$$
\begin{array}{lcllcl}
Cons_1 & \stackrel{def}{=} & (get, r_g).Cons_2 & Prod_1 & \stackrel{def}{=} & (make, r_m).Prod_2 \\
Cons_2 & \stackrel{def}{=} & (cons, r_c).Cons_1 & Prod_2 & \stackrel{def}{=} & (put, r_p).Prod_1 \\
Buf_2 & \stackrel{def}{=} & (get, \top).Buf_1 & Buf_1 & \stackrel{def}{=} & (get, \top).Buf_0 + (put, \top).Buf_2 \\
Buf_0 & \stackrel{def}{=} & (put, \top).Buf_1 & Sys & \stackrel{def}{=} & Cons_1 \underset{\{get\}}{\bowtie} Buf_2 \underset{\{put\}}{\bowtie} Prod_1
\end{array}
$$

$$
\frac{\dfrac{Prod_1 \xrightarrow{(make, r_m)} Prod_2}{Cons_1 \underset{\{get\}}{\bowtie} Buf_2 \underset{\{get\}}{\bowtie} Prod_1 \xrightarrow{(make, r_m)} Cons_1 \underset{\{get\}}{\bowtie} Buf_2 \underset{\{put\}}{\bowtie} Prod_2}}{Sys \xrightarrow{(make, r_m)} Cons_1 \underset{\{get\}}{\bowtie} Buf_2 \underset{\{put\}}{\bowtie} Prod_2}
$$

Summarising, the following transitions were found:

$$
Sys \xrightarrow{(get, r_g)} Cons_2 \underset{\{get\}}{\bowtie} Buf_1 \underset{\{put\}}{\bowtie} Prod_1
$$

$$
Sys \xrightarrow{(make, r_m)} Cons_1 \underset{\{get\}}{\bowtie} Buf_2 \underset{\{put\}}{\bowtie} Prod_2
$$

Consumer/Producer in PEPA

$$
\begin{array}{rclrcl}
Cons_1 & \stackrel{def}{=} & (get, r_g).Cons_2 & Prod_1 & \stackrel{def}{=} & (make, r_m).Prod_2 \\
Cons_2 & \stackrel{def}{=} & (cons, r_c).Cons_1 & Prod_2 & \stackrel{def}{=} & (put, r_p).Prod_1 \\
Buf_2 & \stackrel{def}{=} & (get, \top).Buf_1 & Buf_1 & \stackrel{def}{=} & (get, \top).Buf_0 + (put, \top).Buf_2 \\
Buf_0 & \stackrel{def}{=} & (put, \top).Buf_1 & Sys & \stackrel{def}{=} & Cons_1 \underset{\{get\}}{\bowtie} Buf_2 \underset{\{put\}}{\bowtie} Prod_1
\end{array}
$$

Popping $Cons_2 \underset{\{get\}}{\bowtie} Buf_1 \underset{\{put\}}{\bowtie} Prod_1$ off the stack,

$$
\frac{Cons_2 \xrightarrow{(cons, r_c)} Cons_1}{Cons_2 \underset{\{get\}}{\bowtie} Buf_1 \underset{\{put\}}{\bowtie} Prod_1 \xrightarrow{(cons, r_c)} Cons_1 \underset{\{get\}}{\bowtie} Buf_1 \underset{\{put\}}{\bowtie} Prod_1},
$$

$$
\frac{Prod_1 \xrightarrow{(make, r_m)} Prod_2}{Cons_2 \underset{\{get\}}{\bowtie} Buf_1 \underset{\{put\}}{\bowtie} Prod_1 \xrightarrow{(make, r_m)} Cons_2 \underset{\{get\}}{\bowtie} Buf_1 \underset{\{put\}}{\bowtie} Prod_2}.
$$

Consumer/Producer in PEPA

$$
\begin{array}{rclrcl}
Cons_1 & \stackrel{def}{=} & (get, r_g).Cons_2 & Prod_1 & \stackrel{def}{=} & (make, r_m).Prod_2 \\
Cons_2 & \stackrel{def}{=} & (cons, r_c).Cons_1 & Prod_2 & \stackrel{def}{=} & (put, r_p).Prod_1 \\
Buf_2 & \stackrel{def}{=} & (get, \top).Buf_1 & Buf_1 & \stackrel{def}{=} & (get, \top).Buf_0 + (put, \top).Buf_2 \\
Buf_0 & \stackrel{def}{=} & (put, \top).Buf_1 & Sys & \stackrel{def}{=} & Cons_1 \underset{\{get\}}{\bowtie} Buf_2 \underset{\{put\}}{\bowtie} Prod_1
\end{array}
$$

Therefore, we still need to infer transitions for the following processes...

$$
Cons_1 \underset{\{get\}}{\bowtie} Buf_2 \underset{\{put\}}{\bowtie} Prod_2
$$

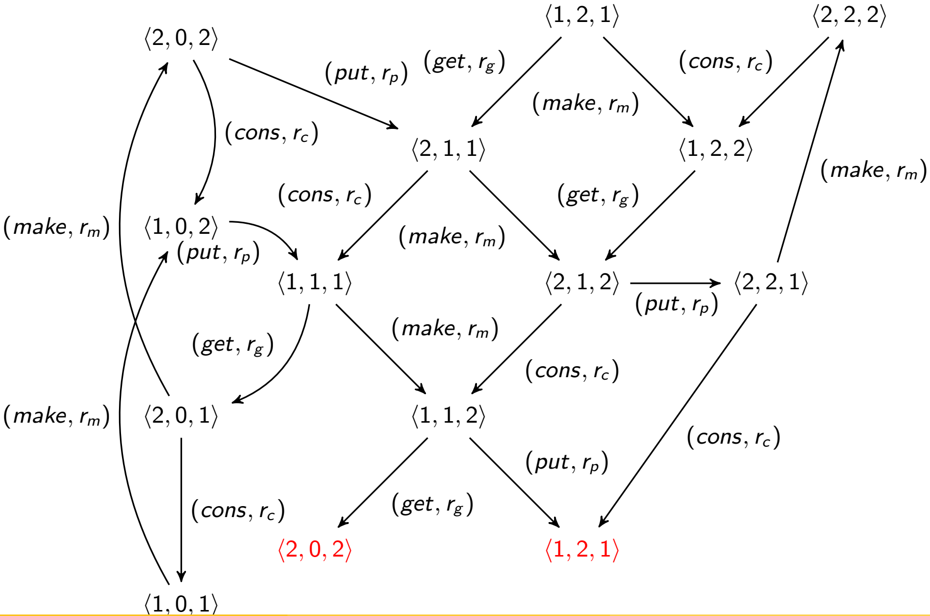
$$
Cons_1 \underset{\{get\}}{\bowtie} Buf_1 \underset{\{put\}}{\bowtie} Prod_1
$$

$$
Cons_2 \underset{\{get\}}{\bowtie} Buf_1 \underset{\{put\}}{\bowtie} Prod_2
$$

...and all those that are found along the way.

Notice that the cooperation structure is fixed across all processes. Thus, we may denote a state by $\langle i, j, k \rangle$ to indicate $Cons_i \underset{\{get\}}{\bowtie} Buf_j \underset{\{put\}}{\bowtie} Prod_k$.

Consumer/Producer in PEPA: Complete Derivation Graph



Consumer/Producer in PEPA: State-Transition Diagram