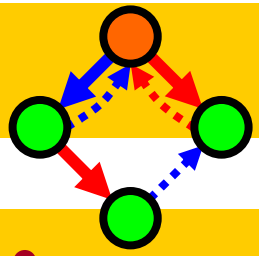
Number rules the universe. Pythagoras.

Artificial Biochemistry



Stochastic Communicating Automata

Luca Cardelli

Microsoft Research

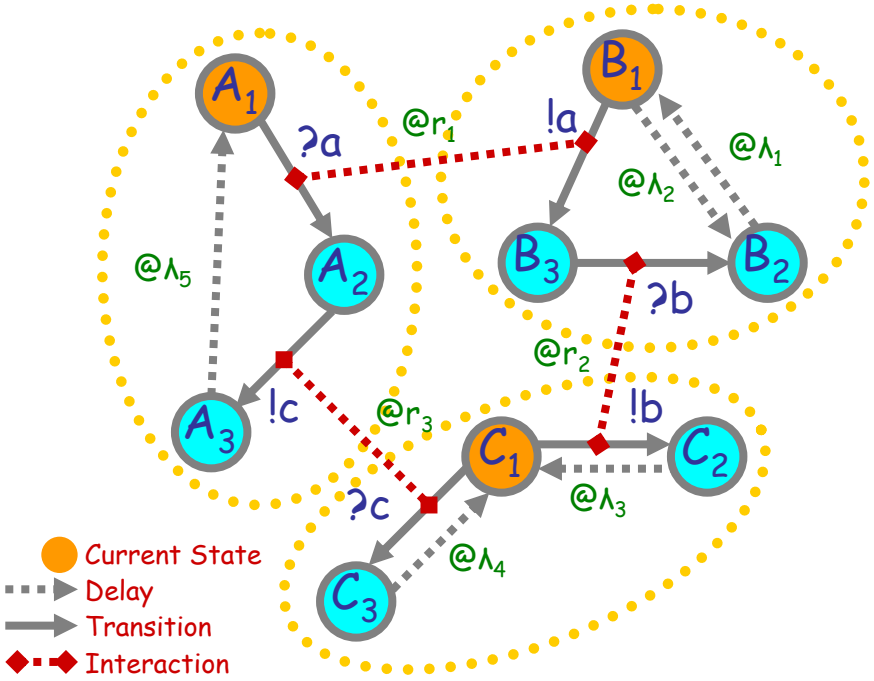
The Microsoft Research - University of Trento
Centre for Computational and Systems Biology

Trento, 2006-05-22..26

www.luca.demon.co.uk/ArtificialBiochemistry.htm

Communicating Automata

Interacting Automata



Communicating automata: a graphical FSA-like notation for "finite state restriction-free π-calculus processes". *Interacting* automata do not even exchange values on communication.

The stochastic version has *rates* on communications, and delays.

$$\begin{aligned}
&\text{new } a@r_1 \\
&\text{new } b@r_2 \\
&\text{new } c@r_3
\end{aligned}\Big\}\ \text{Communication channels}$$

$$\left.\begin{aligned}
A_1 &= ?a;\ A_2 \\
A_2 &= !c;\ A_3 \\
A_3 &= @\lambda_5;\ A_1
\end{aligned}\right\}$$

$$\left.\begin{aligned}
B_1 &= @\lambda_2;\ B_2 + !a;\ B_3 \\
B_2 &= @\lambda_1;\ B_1 \\
B_3 &= ?b;\ B_2
\end{aligned}\right\}$$

$$\left.\begin{aligned}
C_1 &= !b;\ C_2 + ?c;\ C_3 \\
C_2 &= @\lambda_3;\ C_1 \\
C_3 &= @\lambda_4;\ C_2
\end{aligned}\right\}$$

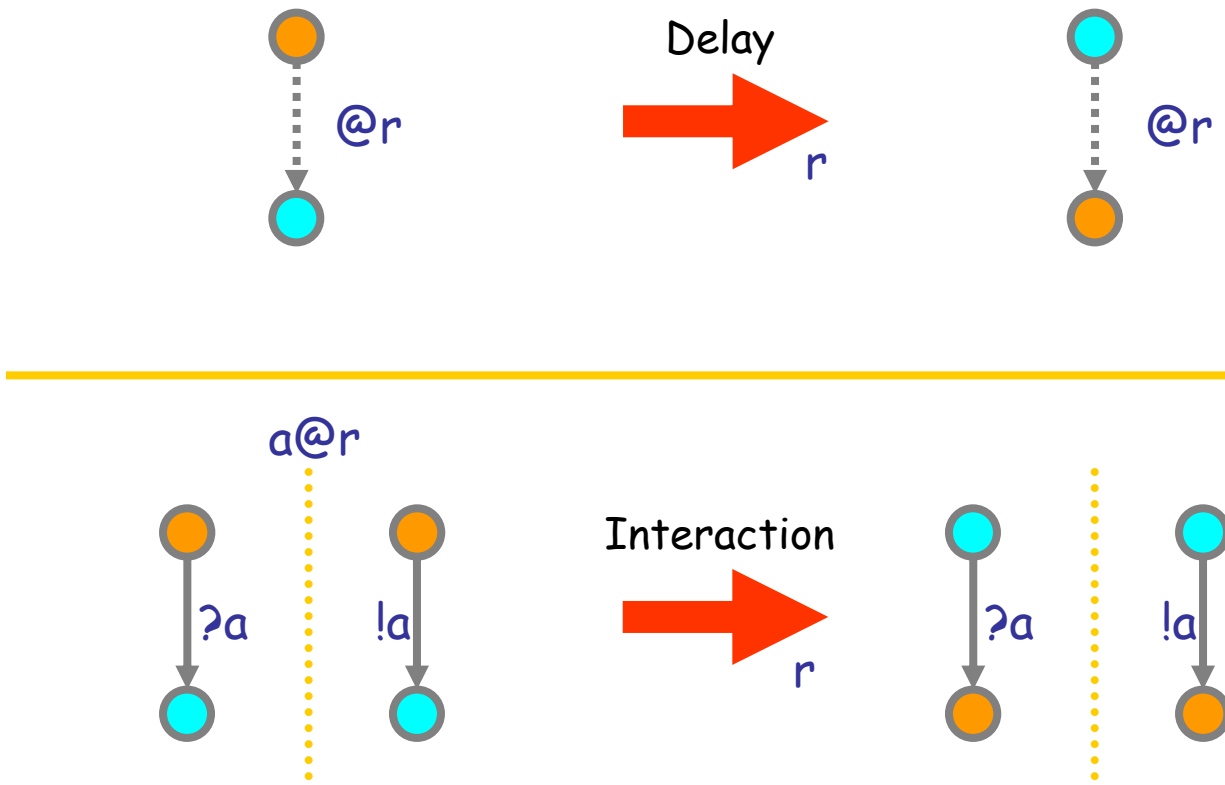
Automata

$$A_1 \mid B_1 \mid C_1\ \Big\}\ \text{The system and initial state}$$

"Finite state" means: no composition or restriction inside recursion.

Analyzable by standard Markovian techniques, by first computing the "product automaton" to obtain the underlying finite Markov transition system. [Buchholz]

Interacting Automata Transition Rules



Current State
Delay
Transition

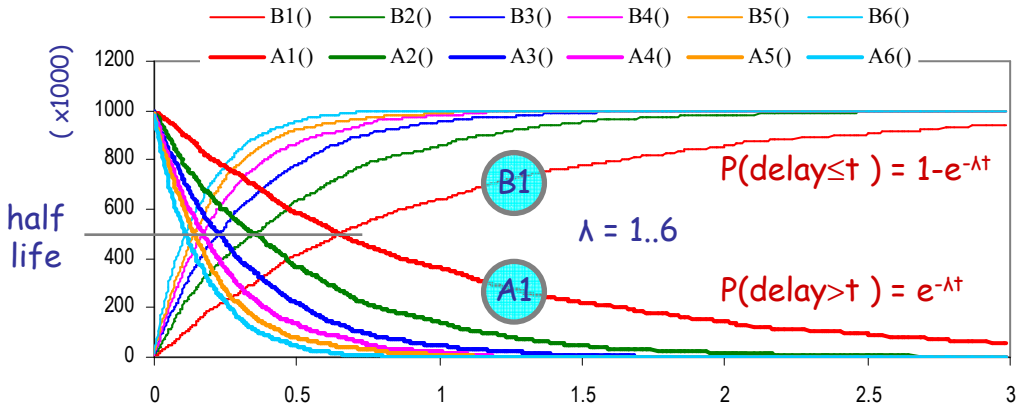
Delay Automata

Delay

$A = @\lambda;B$

$[A]^{\bullet} = -\lambda\,[A]$ [P] is the number of P's

$[B]^{\bullet} = \lambda\,[A]$

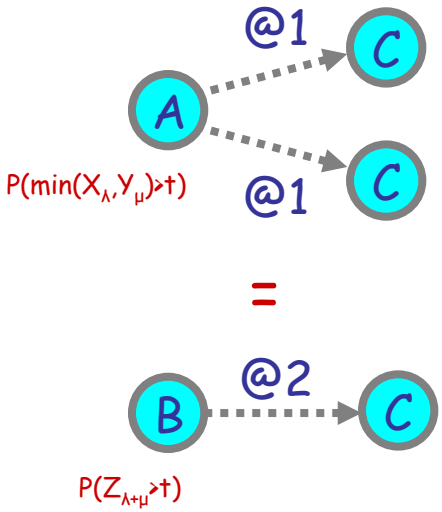


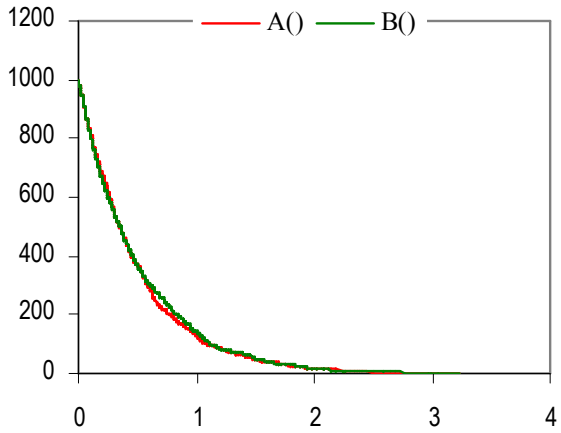

```
directive sample 3.0
directive plot B1(); B2(); B3(); B4(); B5(); B6();
  A1(); A2(); A3(); A4(); A5(); A6()

let A1() = delay@1.0; B1()   and B1() = ()
let A2() = delay@2.0; B2()   and B2() = ()
let A3() = delay@3.0; B3()   and B3() = ()
let A4() = delay@4.0; B4()   and B4() = ()
let A5() = delay@5.0; B5()   and B5() = ()
let A6() = delay@6.0; B6()   and B6() = ()

run 100 of (A1() | A2() | A3() | A4() | A5()| A6())
```

Choice

$$@\lambda;C + @\mu;C = @(\lambda+\mu);C$$

The only extra law of *Strong Markovian Bisimulation* (Lumpability)






```
directive sample 4.0 10000
directive plot A(); B()

let A() = do delay@1.0 or delay@1.0
and B() = delay@2.0

run 1000 of (A() | B())
```

This is the **min of two random variables** (i.e. a duration which is the min of two durations). In the case of exponential distributions, the min is another exponential distribution.

Decay by two or more processes
A quantity may decay via two or more different processes simultaneously. These processes may have different probabilities of occurring, and thus will occur at different rates with different half-lives. For instance, in the case of two simultaneous decay processes, the decay of the quantity *N* is given by:

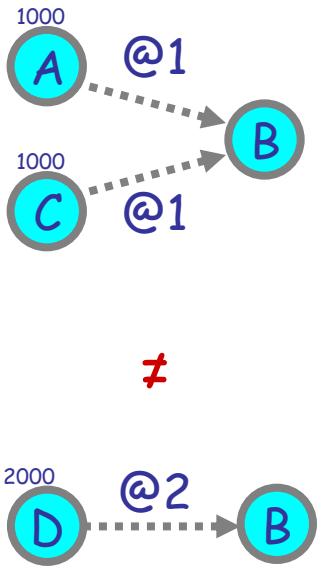
$$N(t) = N_0 e^{-\lambda_1 t} e^{-\lambda_2 t} = N_0 e^{-(\lambda_1+\lambda_2)t}$$

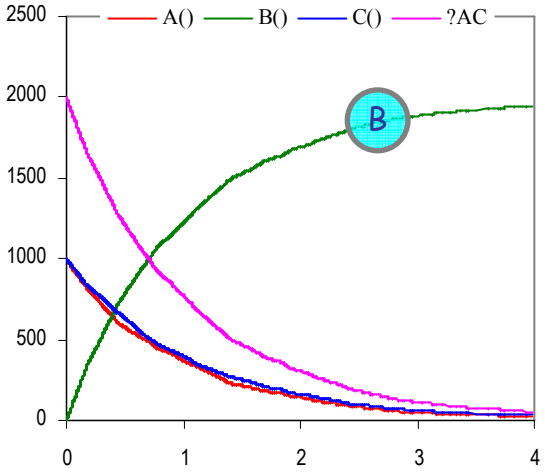
Hence we can sum the rates

Exponential distributions are closed under choice.

Join

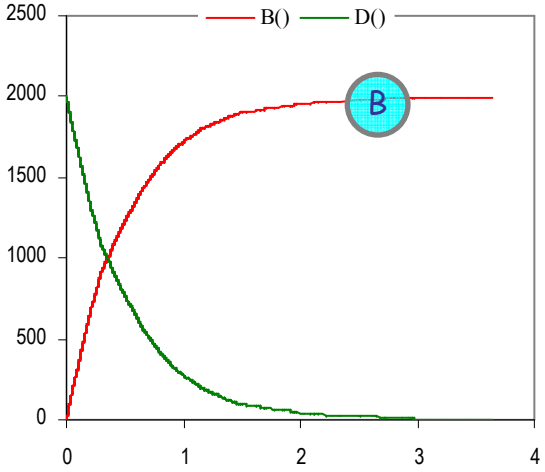
$$@\lambda;B \mid @\mu;B \neq @(\lambda+\mu);\ B$$





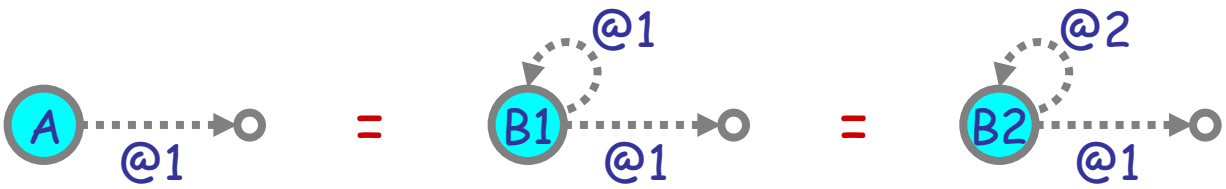

```
directive sample 4.0 10000
directive plot A(); B(); C(); ?AC
new AC@1.0: chan

let A() = do delay@1.0; B() or ?AC
and C() = do delay@1.0; B() or ?AC
and B() = ()

run 1000 of (A() | C())
```

Hmm, so what is this B distribution?
(wait until later)

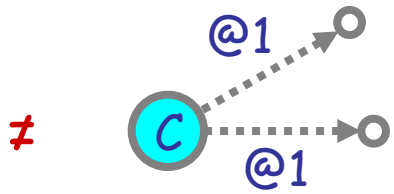



```
directive sample 4.0 10000
directive plot B(); D()

let D() = delay@2.0; B()
and B() = ()

run 2000 of D()
```

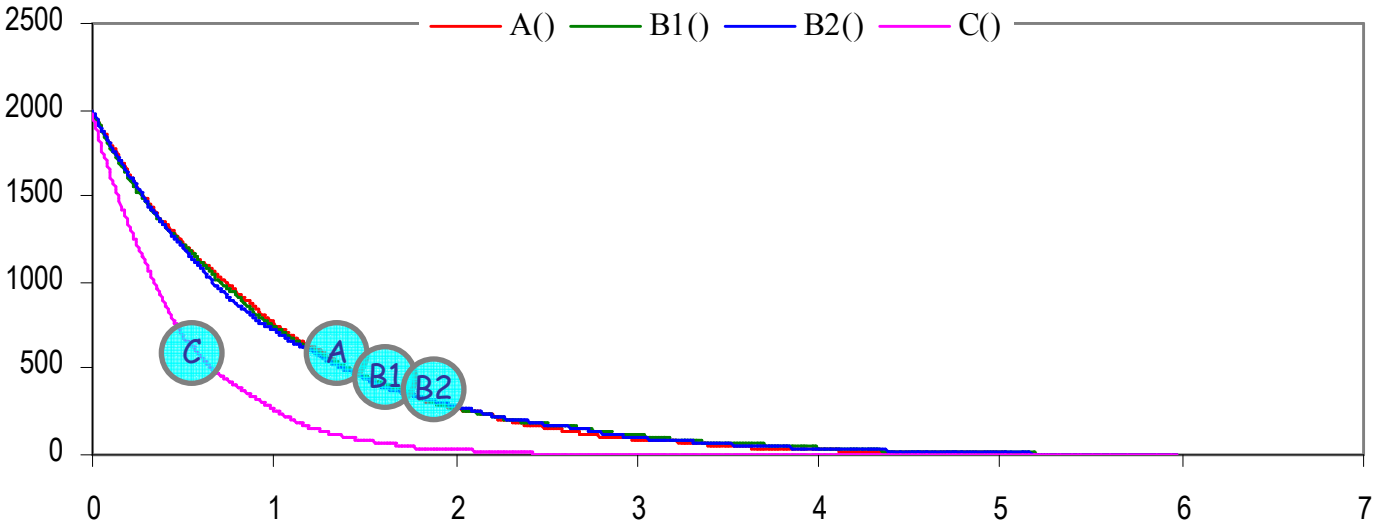
Idle Loops



B2 is depleted at rate 3 and replenished at rate 2, hence…

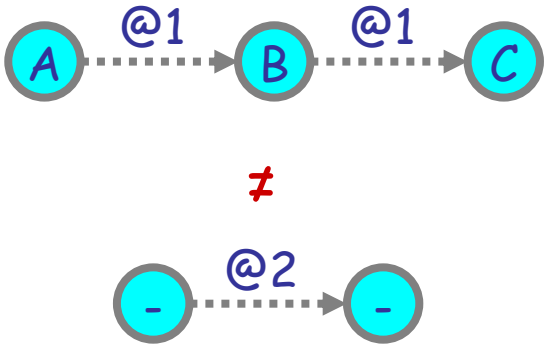


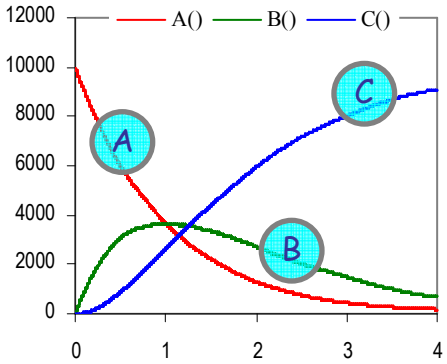

```
directive sample 6.0 1000
directive plot A(); B1(); B2(); C()
let A() = delay@1.0; ()
let B1() = do delay@1.0; B1() or delay@1.0; ()
let B2() = do delay@2.0; B2() or delay@1.0; ()
let C() = do delay@1.0; () or delay@1.0; ()
run 2000 of (A() | B1() | B2() | C())
```



Sequence

@λ; @μ; C = ??






```
directive sample 4.0 10000
directive plot A(); B(); C()

let A() = delay@1.0; B()
and B() = delay@1.0; C()
and C() = ()

run 1000 of A()
```

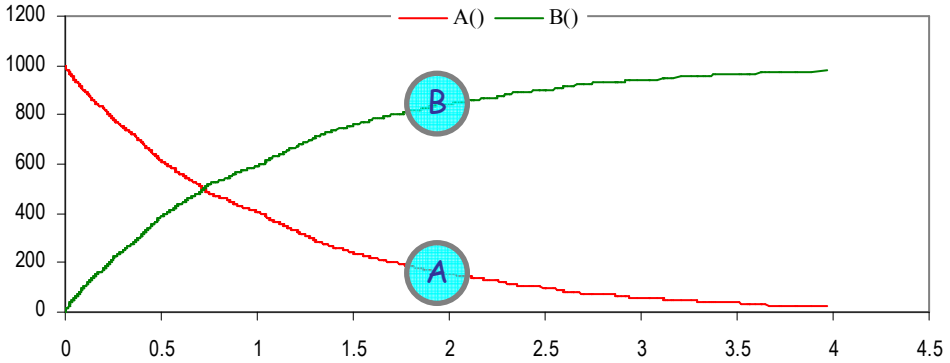
This is the **sum of two random variables** (i.e. a duration which is the sum of two durations). In the case of exponential distributions, the (discrete) sum B is an Erlang distribution.

On the other hand, this means that networks of exponential distributions can express more (much more) than just exponential distributions. (Can densely approximate any probability distribution.)

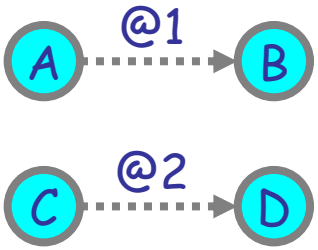
Exponential distributions are **not** closed under sequence.

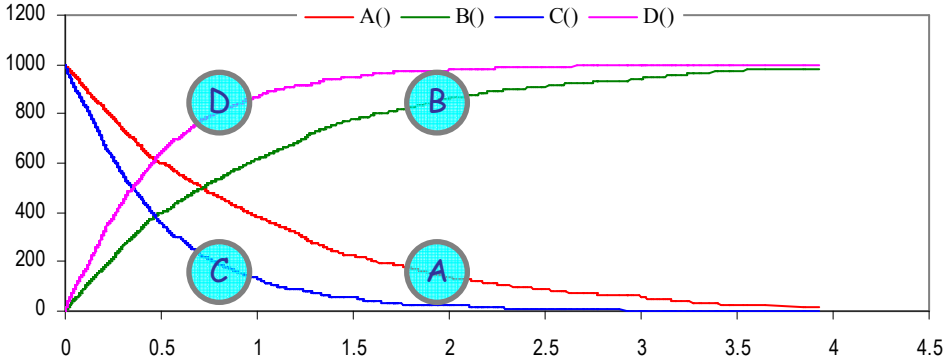
Unbounded Concurrency

$$A \xrightarrow{@\wedge} B \;\Rightarrow\; A|C \xrightarrow{@\wedge} B|C$$




```
directive sample 4.0 10000
directive plot A(); B()

let A() = delay@1.0; B()
and B() = ()

run 1000 of A()
```



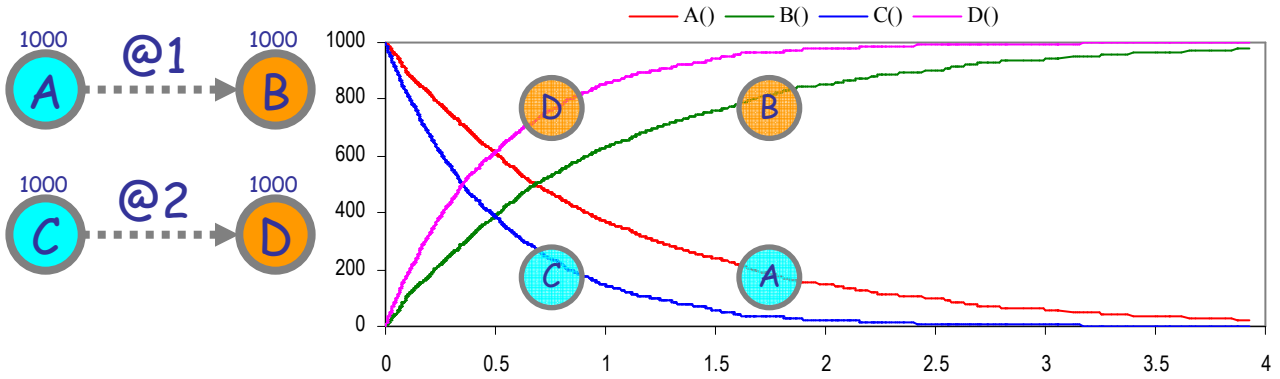



```
directive sample 4.0 10000
directive plot A(); B(); C(); D()

let A() = delay@1.0; B()
and B() = ()

let C() = delay@2.0; D()
and D() = ()

run 1000 of (A() | C())
```

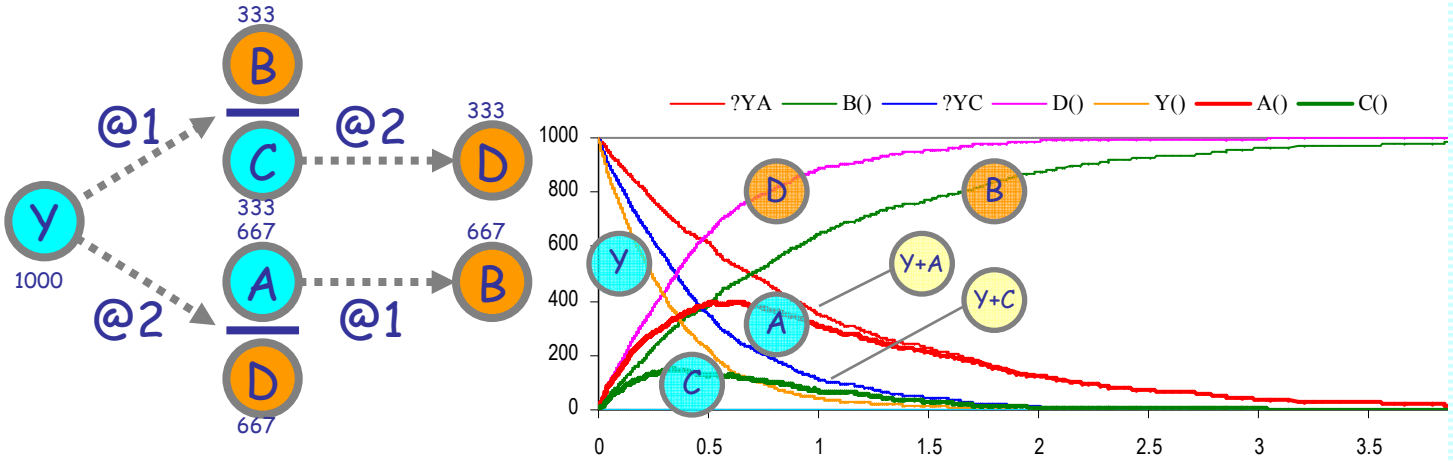
No side effect on A or B.
(E.g. they don't get "slower" because something else is running!)

Asynchronous Interleaving

$$@\lambda;B \mid @\mu;D = @\lambda;(B \mid @\mu;D) + @\mu;(@\lambda;B \mid D)$$




```
directive sample 4.0 10000
directive plot A(); B(); C(); D()

let A() = delay@1.0; B()
and B() = ()

let C() = delay@2.0; D()
and D() = ()

run 1000 of (A() | C())
```



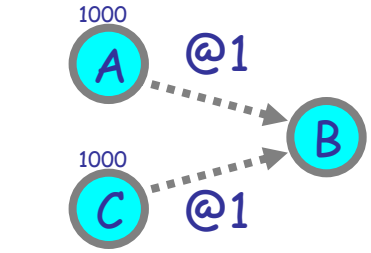

```
directive sample 4.0 10000
directive plot
   ?YA; B(); ?YC; D(); Y(); A(); C()
new YA@1.0:chan new YC@1.0:chan

let A() = do delay@1.0; B() or ?YA
and B() = ()

let C() = do delay@2.0; D() or ?YC
and D() = ()

let Y() =
   do delay@1.0; (B() | C())
   or delay@2.0; (A() | D())
   or ?YA or ?YC

run 1000 of Y()
```

N.B. just to talk about this law, we need to exceed the automata framework and move to process algebra in order to have states that *split*.

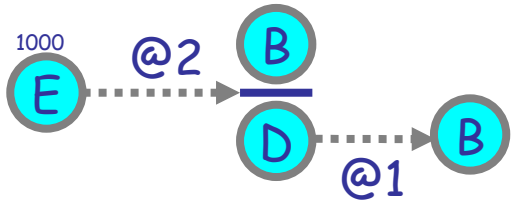
Amazingly, the B's and the D's from the two branches sum up to exponential distributions

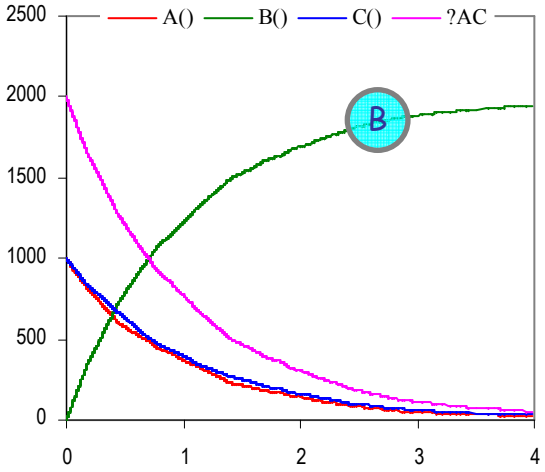
Join Again

@λ;B | @λ;B
= @λ;(B | @λ;B) + @λ;(@λ;B | B) by Interleaving
= @2λ;(B | @λ;B) by Choice

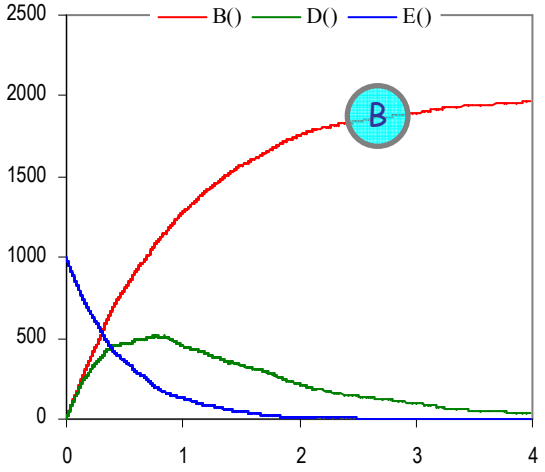


=






```
directive sample 4.0 10000
directive plot A(); B(); C(); ?AC
new AC@1.0: chan

let A() = do delay@1.0; B() or ?AC
and C() = do delay@1.0; B() or ?AC
and B() = ()

run 1000 of (A() | C())
```

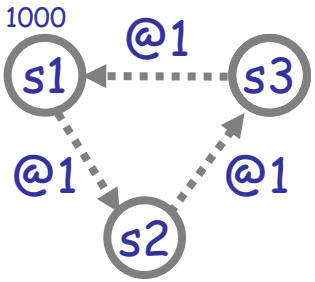



```
directive sample 4.0 10000
directive plot B(); D(); E()

let E() = delay@2.0; (B() | D())
and D() = delay@1.0; B()
and B() = ()

run 1000 of E()
```

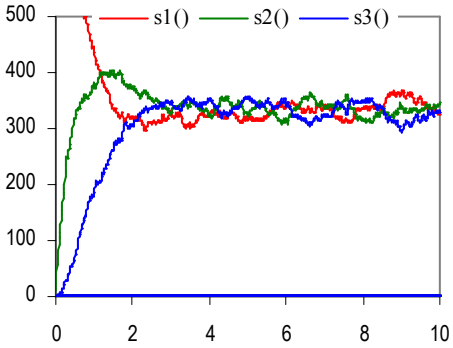
That only works if the two delays are equal;
in general, by Interleaving:

@λ;B | @μ;B = @λ;(B | @μ;B) + @μ;(@λ;B | B)

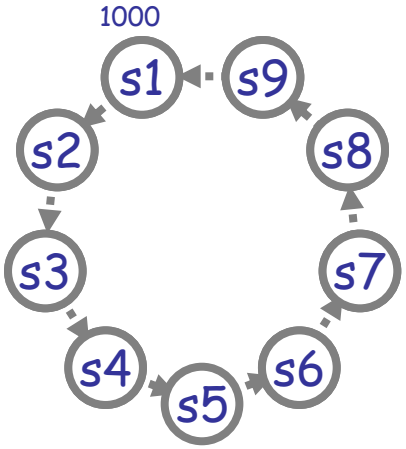
Loops



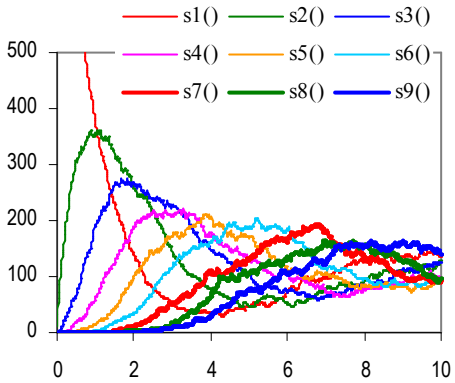
3-loop




```
directive sample 10.0 1000
directive plot s1(); s2(); s3()

let s1() = delay@1.0; s2()
and s2() = delay@1.0; s3()
and s3() = delay@1.0; s1()

run 1000 of s1()
```



9-loop



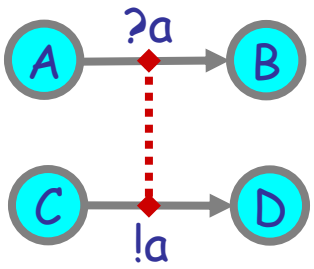

```
directive sample 10.0 1000
directive plot s1(); s2(); s3(); s4();
s5(); s6(); s7(); s8(); s9()

let s1() = delay@1.0; s2()
and s2() = delay@1.0; s3()
and s3() = delay@1.0; s4()
and s4() = delay@1.0; s5()
and s5() = delay@1.0; s6()
and s6() = delay@1.0; s7()
and s7() = delay@1.0; s8()
and s8() = delay@1.0; s9()
and s9() = delay@1.0; s1()

run 1000 of s1()
```

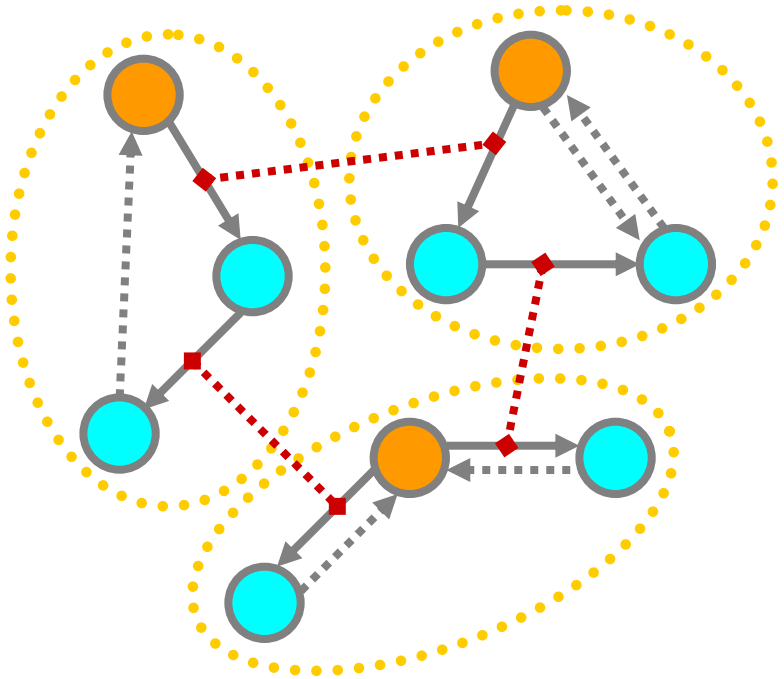
Interacting Automata

From Delay to Interaction

Delay

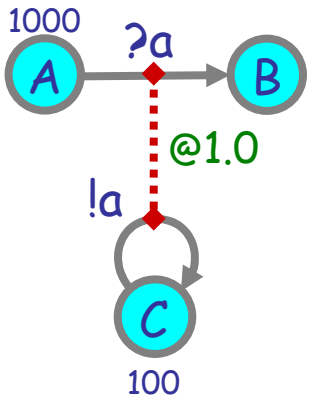
Interaction



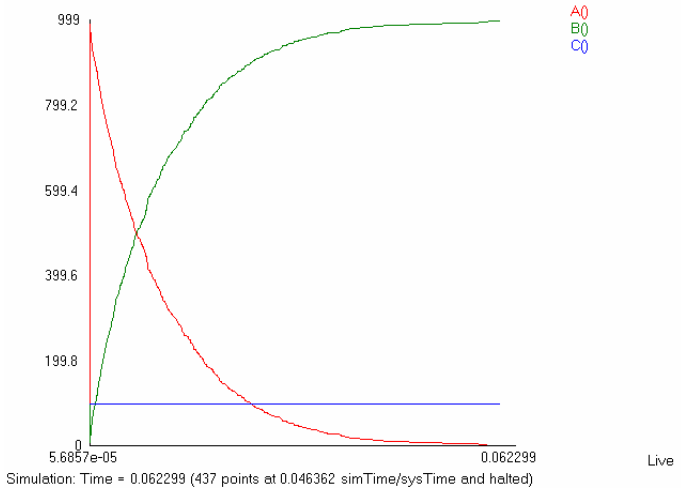


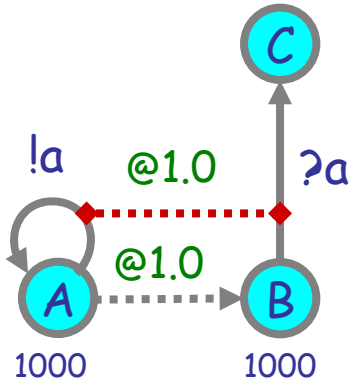
: transitions that have a rate

Examples

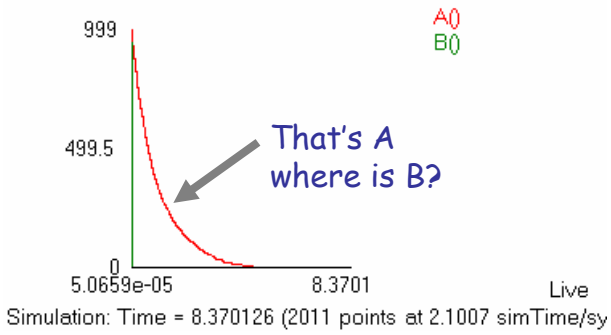


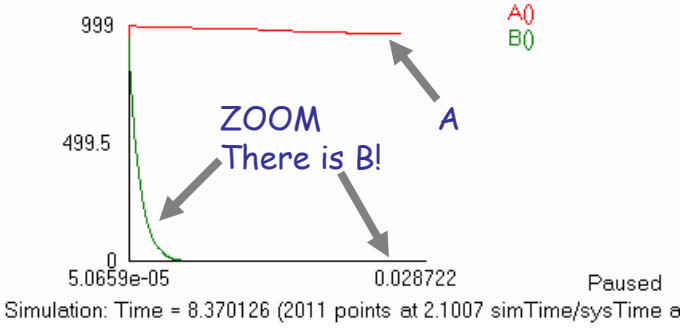

```
directive sample 0.05 1000
directive plot A(); B(); C()

new a@1.0: chan

let A() = ?a; B()
and B() = ()
and C() = !a; C()

run (1000 of A() | 100 of C())

(* try 1,10,1000 of C() *)
```






```
directive sample 0.05 1000
directive plot A(); B()

new a@1.0: chan

let A() =
  do !a; A() or delay@1.0; B()
and B() = ?a; C()
and C() = ()

run (1000 of A() | 1000 of B())
```





B's turn into C's much faster than A's turn into B's.
Now we will spend time understanding
why that happens and what it means.

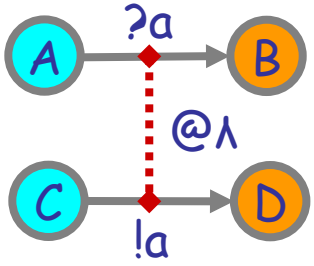
The Rate of What?

- In chemistry:
  - Each reaction involves **2** molecules, and each reaction has a **rate**. Rates belongs to **reactions**. Molecules do *not* have rates.

Rates belong to **channels** not to actions!

- In process algebras:
  - Should rates belong to:
    - each individual action? only outputs? delays only?
  - The rate of a synchronization of two actions should be the:
    - max? product? undefined if different? infinite (except for delays)?
  - All that has been tried.

- We go back to chemistry
  - Rates belong to **channels**. (This is called the "biochemical" stochastic π-calculus by Priami-Regev-Shapiro-Silverman)

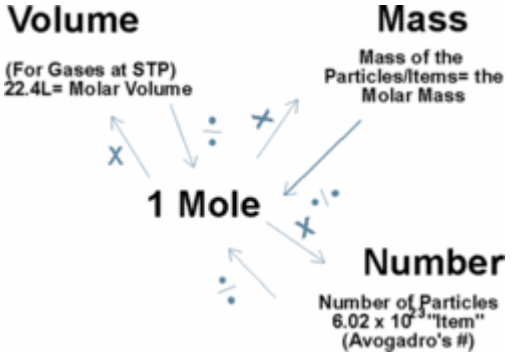


- Issues:
  - Multiple activities on the same channel (concentrations of molecules involved in a reaction: mass action law of chemistry).
  - Choices between different channels (molecules involved in multiple reactions: still standard chemistry).
  - In biochemistry, rates of homodimerization (a molecule can interact with a copy of itself, but not with itself).

Chemical Reaction Rates

http://en.wikipedia.org/wiki/Reaction_rate

The **reaction rate** for a reactant or product in a particular reaction is defined as the amount (in moles or mass units) per unit time per unit volume that is formed or removed.



Reaction rate is often expressed in the units mol/Ls (where 1 mole is a dimensionless constant equal to the Avogadro number).

Concentration is mol/L , hence **rate** is concentration/s.

One also often sees μM (one millionth of a mole *per liter*).That's a *concentration*, not a mole number.

The Concentration of What?

If P is a process (state), then:

[P] is the function that at time t gives the quantity of P

N.B.; Avogadro's number (~6.022×10^{23}) relates concentration to quantity of molecules. Hence we usually identify concentration=quantity. Remember, though that increasing quantity then means increasing concentration, not volume, of solution.

This notion of [P] assumes some way of counting P's, i.e. some way of telling when two P are equal, i.e. a congruence relation on processes.

A simpler option (and what is actually done in SPiM) is never to count "processes" but rather to count "offers of communication" that processes are performing, i.e. active actions (a.k.a. "barbs").

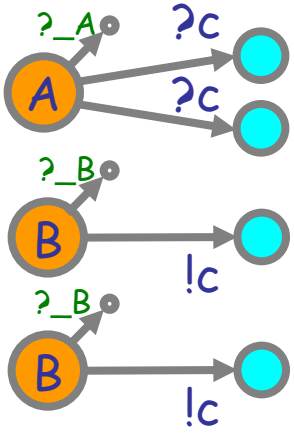
If b is a barb ("?c" or "!c") then:

[b] is the function that at time t gives the quantity of b

If we see P as an automaton, this is very easy to arrange: add a barb to the "current" state of the automaton corresponding to P (e.g. a unique ?_P that nobody ever uses). Then we set [P] = [?_P].

We can use barbs to count processes.

Concentrations belong to **actions** not to processes!



$[?c]_{now} = 2$
$[!c]_{now} = 2$
$[A]_{now} = [?\_A]_{now} = 1$
$[B]_{now} = [?\_B]_{now} = 2$

The Rate of Change of Concentrations

Derivative is an operator that maps continuous functions to continuous functions.
It is defined [Newton] as the higher-order function:

derivative: $(R \to R) \to (R \to R) = \lambda f.\ \lambda x.\ \lim(h \to 0).\ (f(x+h)-f(x))/h$

$f^{\bullet}$ stands for derivative(f) ($\dot{f}$ [Newton], f' [Lagrange])

A differential equation

$[P]^{\bullet} = \ldots$

represents the rate of change of the number (a.ka. concentration) of P's over time.

In general we may have a system of differential equations among concentrations:

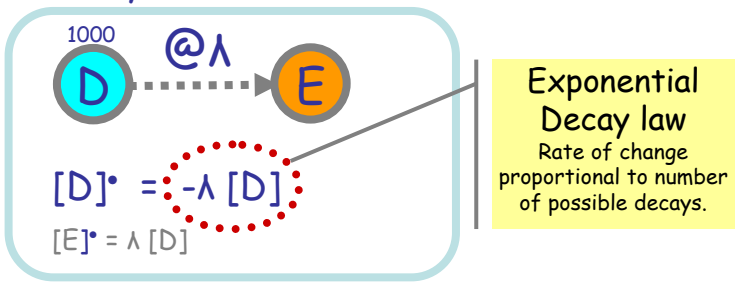
$[A]^{\bullet} = \ldots [A] \ldots [B] \ldots [C] \ldots$
$[B]^{\bullet} = \ldots [A] \ldots [B] \ldots [C] \ldots$
$[C]^{\bullet} = \ldots [A] \ldots [B] \ldots [C] \ldots$

which may be hard to solve symbolically, in which case we will have to solve it numerically (for some specific values of initial concentrations).

The Law of Mass Interaction

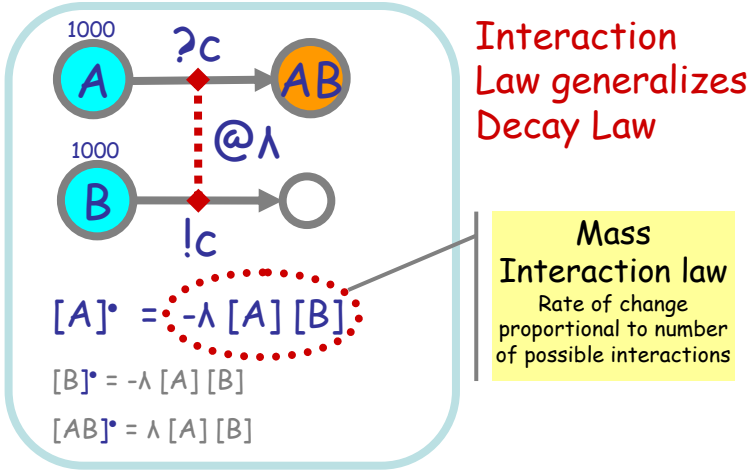
The speed of interaction† is proportional to the number of *possible interactions*.

Decay



Exponential Decay law
Rate of change proportional to number of possible decays.

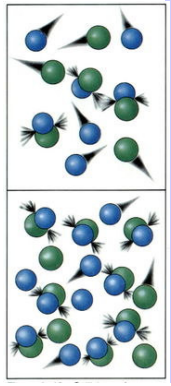
Mass interaction



Interaction Law generalizes Decay Law

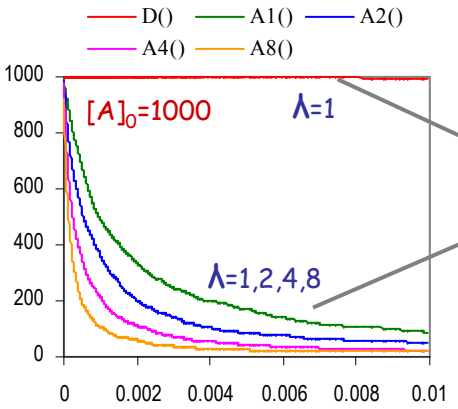
Mass Interaction law
Rate of change proportional to number of possible interactions

† speed of interaction (formally definable)
= number of interactions over time
not proportional to the number of interacting processes!
[P] is the number of processes P (this is informal; it is only meaningful for a set of processes offering a given action, but a set of such processes can be counted and plotted)



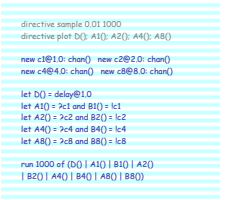
Chemical Law of Mass Action
http://en.wikipedia.org/wiki/Chemical_kinetics
The **speed** of a chemical reaction is proportional to the **activity** of the reacting substances.
Activity = concentration, for well-stirred aqueous medium
Concentration = number of moles per liter of solution
Mole = 6.022141×10^{23} particles

It's not an opinion, it's the Law!



decay

interaction (much faster!)



Activity and Speed

stochastic algebras disagree!

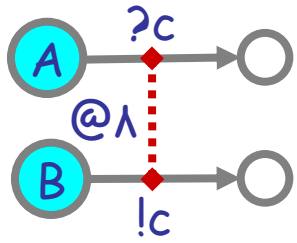
The speed of interaction is proportional to the number of possible interactions.

=

The *activity* (= "concentration") on a channel is the number of *possible interactions* on that channel.

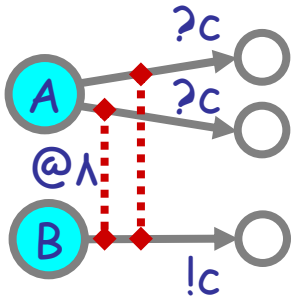
The *speed of interaction* on a channel, is the activity multiplied by the base rate of the channel.

```
directive sample 0.01 10000
directive plot A1(); A2(); A3()

new c1@1.0:chan
new c2@1.0:chan
new c3@1.0:chan

let A1() = ?c1
and B1() = !c1

let A2() = do ?c2 or ?c2
and B2() = !c2

let A3() = do ?c3 or ?c3
and B3() = do !c3 or !c3

run 1000 of (A1() | B1()
| A2() | B2() | A3() | B3())
```



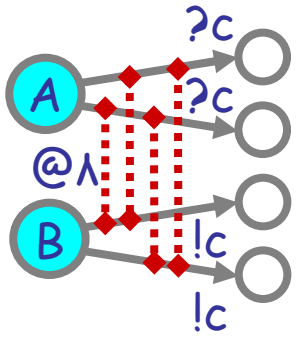
c activity: 1

speed: λ



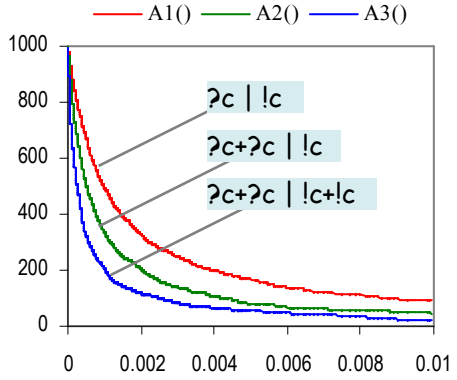
c activity: 2

speed: 2λ



c activity: 4

speed: 4λ

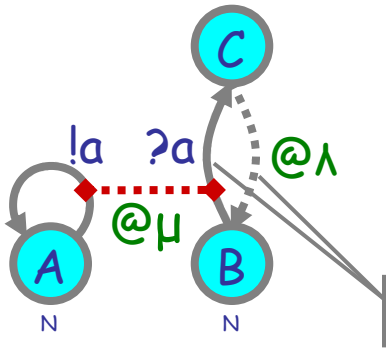


The mass interaction law [Buchholz] [Priami-Regev-Shapiro-Silverman] is compatible with chemistry [Gillespie] and *incompatible* with any other stochastic algebra in the literature! (including [Priami]; see [Hermanns])

Other algebras assign rates to actions, not channels, with speed laws:

- $2\lambda * 2\lambda = 4\lambda^2$
- $\max(2\lambda, 2\lambda) = 2\lambda$ [Goetz]
- $\min(2\lambda, 2\lambda) = 2\lambda$ [Priami]
- $1/(1/(2\lambda)+1/(2\lambda)) = \lambda$ [PEPA]
- $2\lambda * 1 = 2\lambda$ (passive inputs)

The Strength of Populations

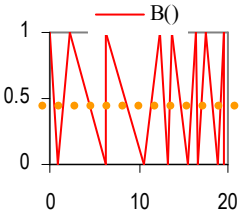


At size 2N, on a shared channel,
μ is N times stronger than λ:
interaction easily wins over delay.

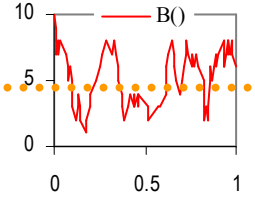
λ-μ fight!

```
directive sample 0.01 1000
directive plot B()

val lam = 1000.0
val mu = 1.0

new a@mu:chan
let A() = !a; A()
and B() = ?a; C()
and C() = delay@lam; B()

run 1000 of (A() | B())
```

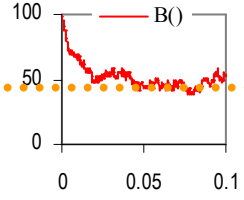
N=1
λ=1
μ=1



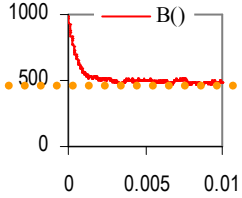
N=10
λ=10
μ=1



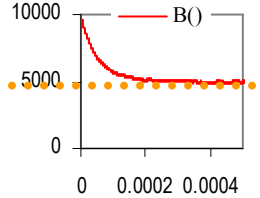
N=100
λ=100
μ=1



N=1000
λ=1000
μ=1



N=10000
λ=10000
μ=1



Equilibrium

Mixed Interaction

The Law of Mixed Interaction

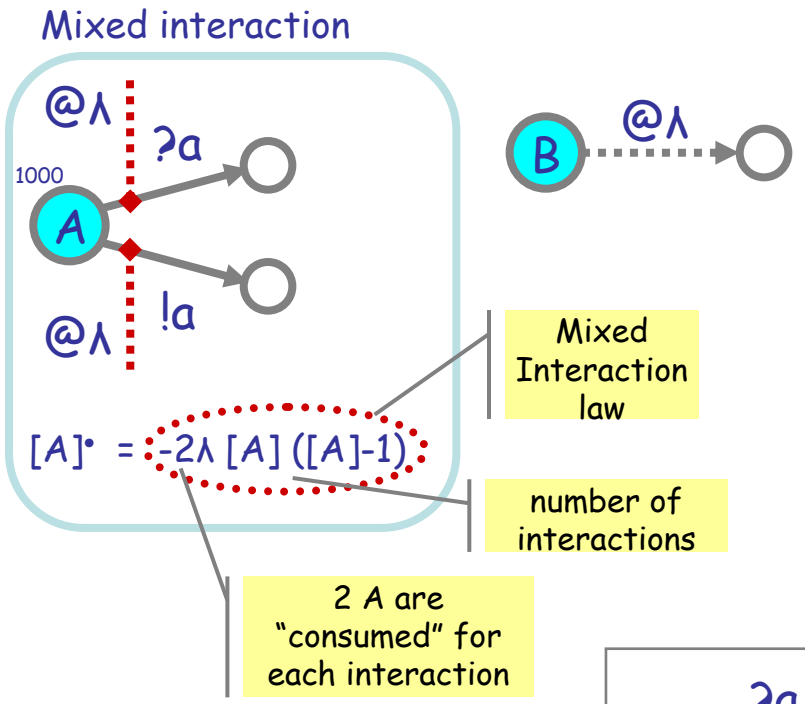
for processes offering both ?a and !a actions

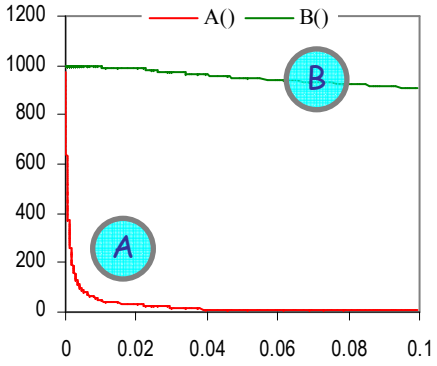
The speed of interaction is proportional to the number of possible interactions.

But a process cannot interact with itself.

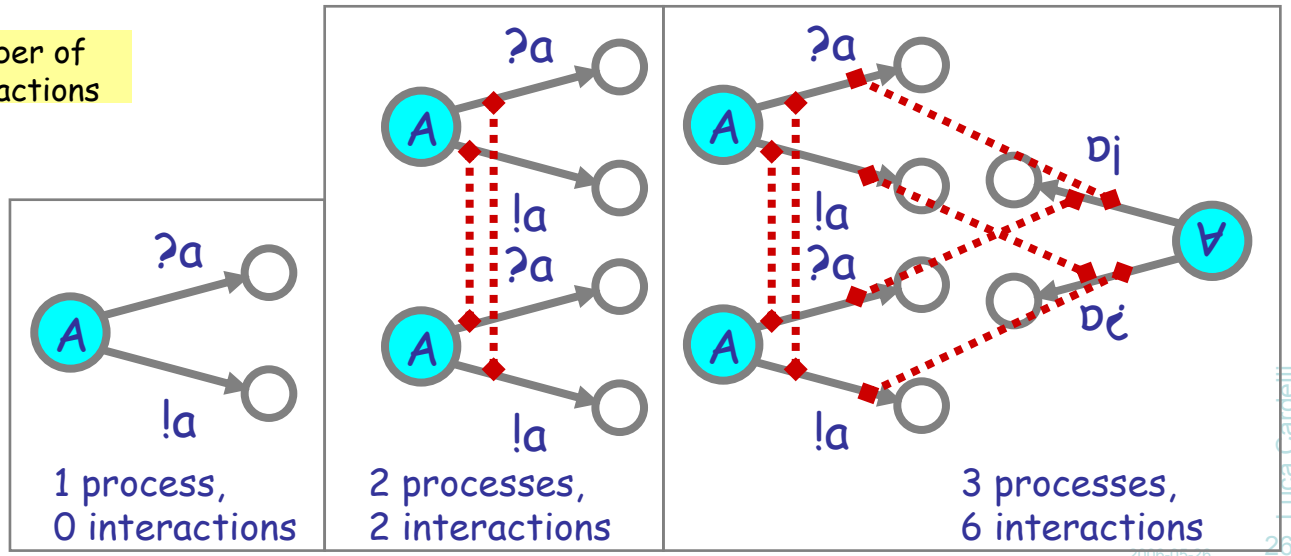
```
directive sample 0.1 10000
directive plot A(); B()

new a@1.0:chan

let A()= do ?a or !a

let B() = delay@1.0

run 1000 of (A() | B())
```

In CCS-style process algebras, not even [Priami-Regev-Shapiro-Silverman] or BioSPI have this law in full generality. It was first worked out by [Block et al (unpub)], and only SPIM implements it.





Note for later: Hence, if we want use 3 such processes to model "the 3 possible collisions between 3 particles", then we need to divide the rate λ by 2, because this model gives 6 interactions, not 3, between 3 processes



Possible Interactions

The speed of interaction is proportional to the number of possible interactions.
(And a process cannot interact with itself.)

Assume each process P is in restricted-sum-normal-form. For each channel x:

In(x,P) = Num of active ?x in P

Out(x,P) = Num of active !x in P

Mix(x,P) = In(x,P)*Out(x,P)

#interactions that cannot happen in a given summation P

In(x) = Sum P of In(x,P)

Out(x) = Sum P of Out(x,P)

Mix(x) = Sum P of Mix(x,P)

total #interactions that cannot happen

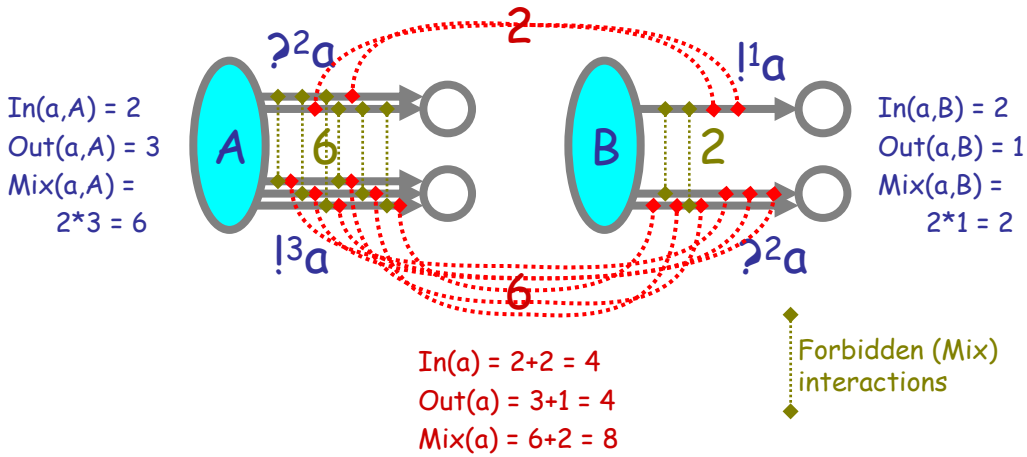
The global Activity on channel x:

$$Act(x) = (In(x)*Out(x))-Mix(x)$$

total cross product of inputs and outputs minus total #interactions that cannot happen

The global speed of interaction on a channel x:

$$speed(x) = Act(x)*rate(x)$$



Act(a) = (In(a) * Out(a)) – Mix(a) = 4*4 – 8 = 8

speed(a) = Act(a)*rate(a) = 8*rate(a)

This is the SPiM Activity computation.

Symmetric Reactions by Mixed Choice

Consider a reaction

$A + B \rightarrow^r C$

which has mass action kinetics:

$[C]^\bullet = r[A][B]$

$[A]^\bullet = [B]^\bullet = -r[A][B]$

because [A][B] is the number of possible collisions. For [A]=1 and [B]=1 we have speed

$r * 1 * 1 = r$

i.e. two single molecules interact at speed r, as expected.

Consider now a symmetric reaction

$A + A \rightarrow^r C$

which has mass action kinetics:

$[C]^\bullet = r[A]([A]-1)\frac{1}{2}$

$[A]^\bullet = -2r[A]([A]-1)\frac{1}{2} = -r[A]([A]-1)$

because $[A]([A]-1)\frac{1}{2}$ is the number of possible collisions (*symmetric* interactions) in [A] molecules!

For [A]=1 we have speed

$r * 1 * 0 * \frac{1}{2} = 0$

since a single molecule does not collide.

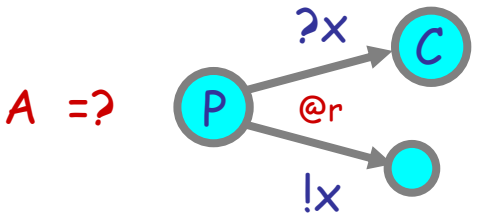
For [A]=2 we have speed

$r * 2 * 1 * \frac{1}{2} = r$

so two single molecules collide at rate r, as before. For [A]=3 we have speed=3, since 3 molecules have 3 possible collisions. Etc. For [A]=n we have as reaction rate (i.e. as C production rate):

$r*n*(n-1)*\frac{1}{2}$

If we model symmetric reactions with mixed choice at rate r, we get the wrong answer:



In(x,P) = 1, Out(x,P) = 1
Mix(x,P) = In(x,P)*Out(x,P) = 1

For n such processes:
In(x) = Sum P of In(x,P) = n*1 = n
Out(x) = Sum P of Out(x,P) = n*1 = n
Mix(x) = Sum P of Mix(x,P) = n*1 = n

The global Activity on channel x:
$Act(x) = (In(x)*Out(x))-Mix(x) = n^2-n = n*(n-1)$

The global speed on channel x:
speed(x) = rate(x)*Act(x) =

$r*n*(n-1)$

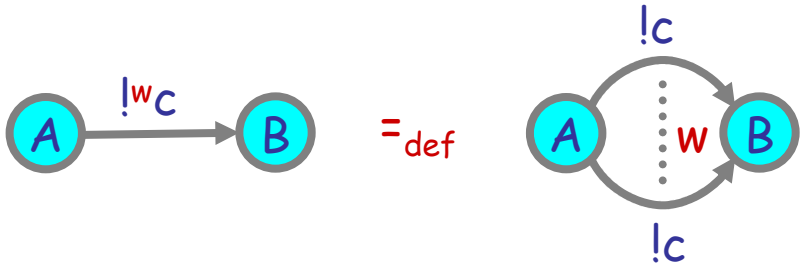
While 1 process has speed=0, as expected, 2 processes have speed=r*2, not r.

We are off by a factor of 2.

The rate of a channel modeling a symmetric reaction must be halved. [Shapiro 2001]

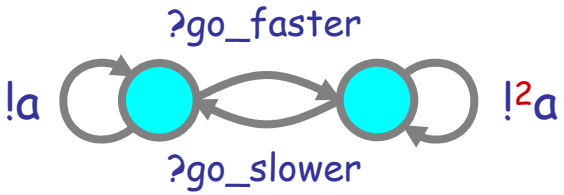
Weighted Actions

Weighted Actions [Buchholz]



Activity = w
(similarly for ?^w c)

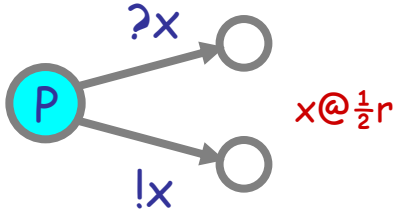
Can generalize w to a real number, and to input-bound variables.

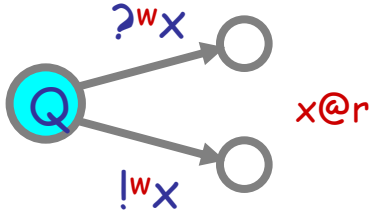


In general, we can allow real numbers as weights, and use them in the activity computation:

In(x,P) = Sum of weights of active ?x in P
Out(x,P) = Sum of weights of active !x in P

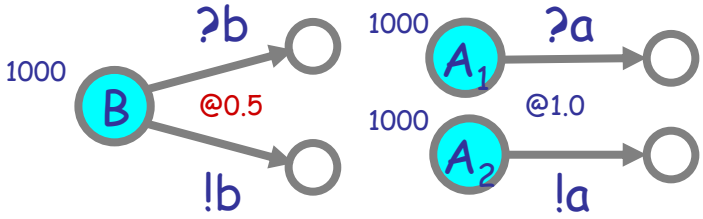
Exercise: Find a weight w such that the following automata interact at the same apparent rate. I.e., instead of "fixing" the rate of symmetric reaction, fix the way they are modeled.



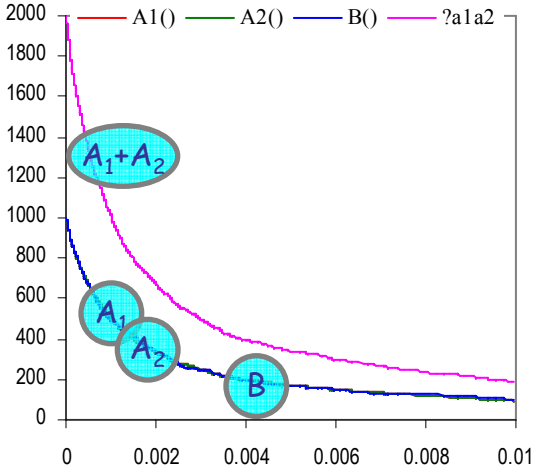


Speed of Symmetric Reactions

Here is a mixed choice interaction representing a symmetric reaction, where the rate has been halved. The B reaction is about as fast as the A reaction, but it uses only half the number of processes.




```
directive sample 0.01 10000
directive plot A1(); A2(); B(); ?a1a2
new a1a2@1.0:chan

new a@1.0:chan
new b@0.5:chan

let A1() = do ?a or ?a1a2
and A2() = do !a or ?a1a2

let B()= do ?b or !b

run 1000 of (A1() | A2())
run 1000 of B()
```



Symmetric reactions use half the materials of binary reactions and are only infinitesimally slower

Cf.: homodimerization (symmetric complexation) is pervasive in biochemistry.

$$[A_i]^\bullet = -1.0*[A_1]*[A_2]$$

$$[B]^\bullet = -2*(1.0*\tfrac{1}{2})*[B]*([B]-1) = -1.0*[B]*([B]-1)$$

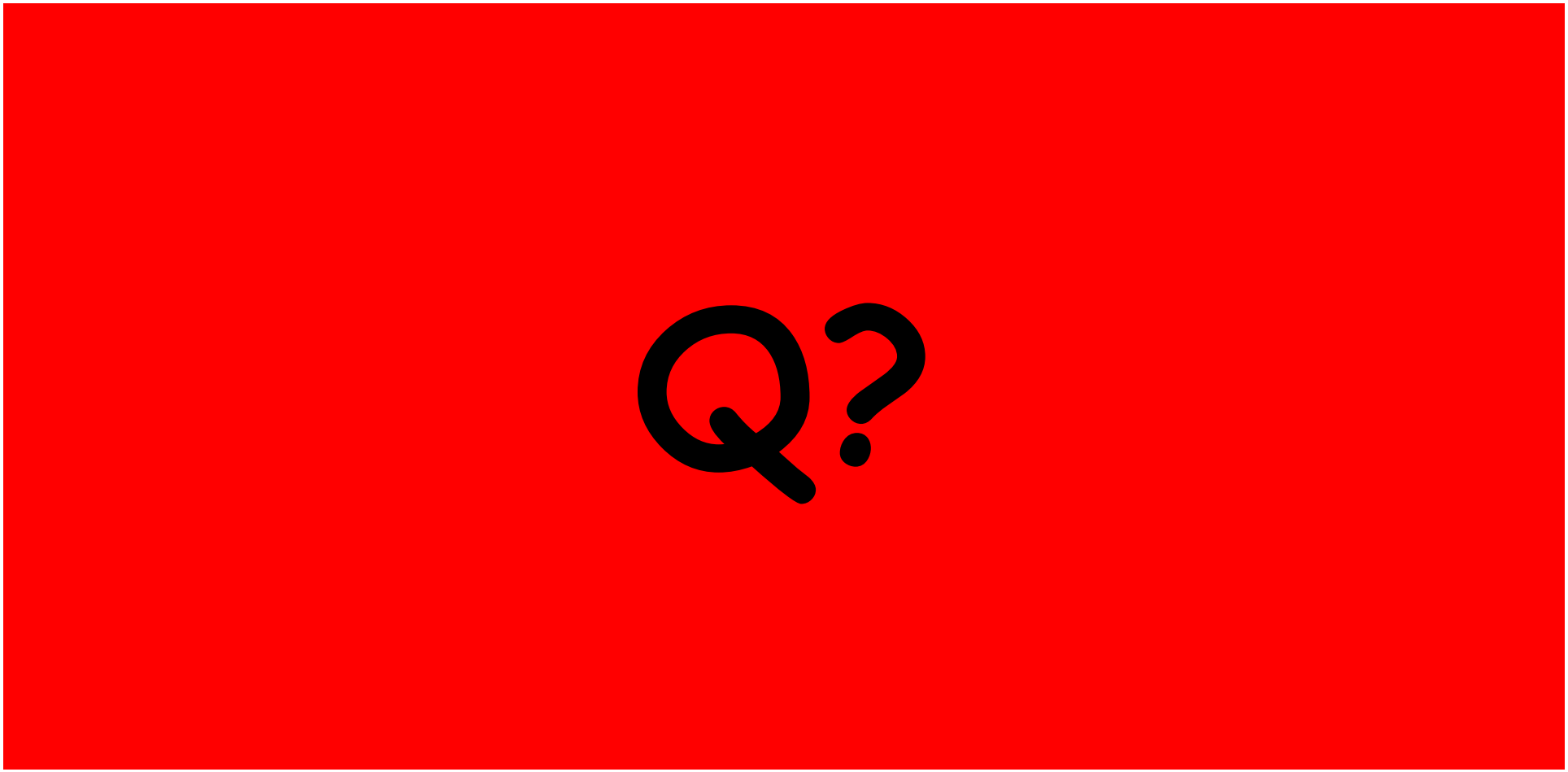
2*B consumed in each reaction

artificially halved rate

missing ½ factor because computed by the *wrong* mixed choice model

Summary

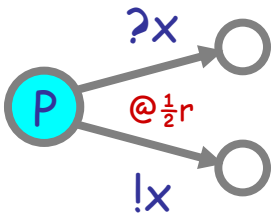
- Delay Automata
  - Governed by the exponential decay law
- Interacting Automata
  - Governed by the mass action law
  - Be careful about symmetric interactions
- The Law of Mass Interaction
  - The speed of interaction is proportional to the number of possible interactions
  - Summarizes all of the above


Q?

Exercise Solution

Symmetric Reactions with Weighted Actions

The rate of a channel modeling a symmetric reaction must be halved.



In(x,P) = Num of active ?x in P = 1
Out(x,P) = Num of active !x in P = 1
Mix(x,P) = In(x,P)*Out(x,P) = 1

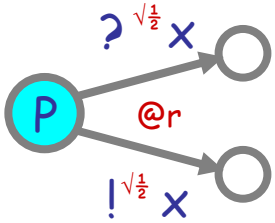
For n such processes:

In(x) = Sum P of In(x,P) = n*1 = n
Out(x) = Sum P of Out(x,P) = n*1 = n
Mix(x) = Sum P of Mix(x,P) = n*1 = n

The global Activity on channel x:
Act(x) = (In(x)*Out(x))-Mix(x) = n^2-n
= n*(n-1)

The global speed on channel x:
speed(x) = rate(x)*Act(x)
= ½*r*n*(n-1)

Or else we can use the *right* mixed choice model, via weighted actions!



In(x,P) = Sum of weights of active ?x in P = √½
Out(x,P) = Sum of weights of active !x in P = √½
Mix(x,P) = In(x,P)*Out(x,P) = ½

For n such processes:

In(x) = Sum P of In(x,P) = n*√½
Out(x) = Sum P of Out(x,P) = n*√½
Mix(x) = Sum P of Mix(x,P) = n*½

The global Activity on channel x:
Act(x) = (In(x)*Out(x))-Mix(x) =
(n*√½)*(n*√½)-(n*½) = (n^2*½)-(n*½) = (n^2-n)*½
= n*(n-1)*½

The global speed on channel x:
speed(x) = rate(x)*Act(x)
= r*n*(n-1)*½

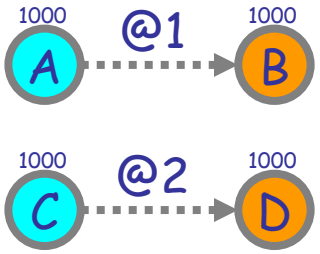
The correct rate for a symmetric reaction.

Appendix
Interleaving NOT

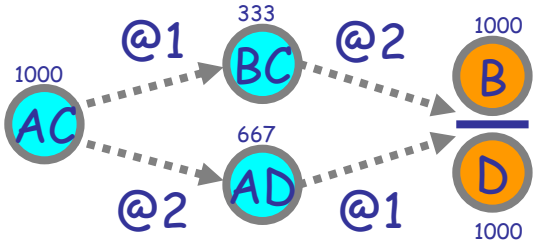
Interleaving NOT

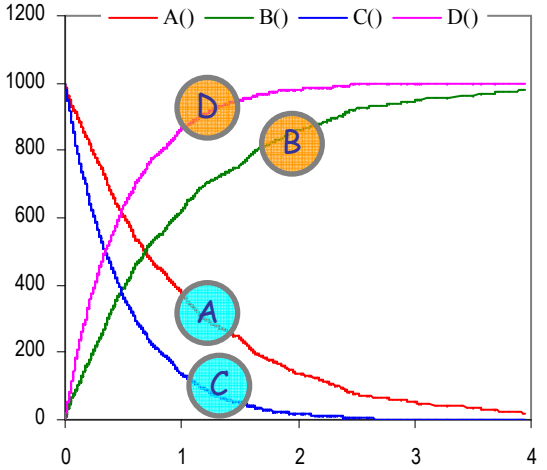
You cannot "*run parallel actions by executing their first step in either order*" !

$$@\lambda;B \mid @\mu;D \neq @\lambda;\ @\mu;\ (B|D) + @\mu;\ @\lambda;\ (B|D)$$

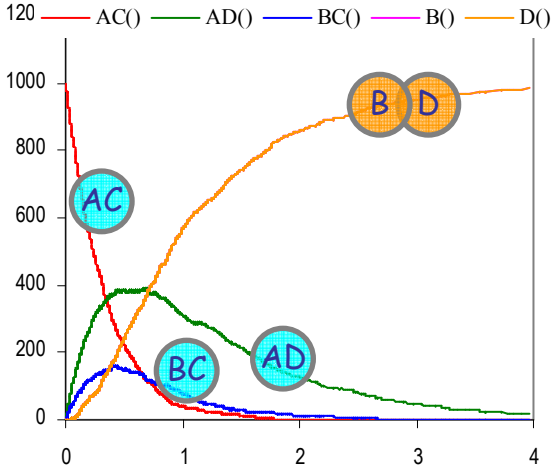


≠






```
directive sample 4.0 10000
directive plot A(); B(); C(); D()

let A() = delay@1.0; B()
and B() = ()

let C() = delay@2.0; D()
and D() = ()

run 1000 of (A() | C())
```




```
directive sample 4.0 10000
directive plot AC(); AD(); BC(); B(); D()

let AC() =
  do delay@1.0; BC()
  or delay@2.0; AD()
and AD() = delay@1.0; (B()|D())
and BC() = delay@2.0; (B()|D())
and B() = ()
and D() = ()

run 1000 of AC()
```

Appendix
Derivatives for Functional Programmers

The Rate of Change

Differentiation for Functional Programmers (like me)

Derivative is an operator that maps continuous functions to continuous functions. It is defined [Newton] as the higher-order function:

derivative = $\lambda f.\ \lambda x.\ \lim(h \to 0).\ (f(x+h)-f(x))/h$

derivative: $(R \to R) \to (R \to R)$

$f^{\bullet}$ stands for derivative(f) ($\dot{f}$ [Newton], f' [Lagrange])

$\sin^{\bullet} = \cos$	a true fact: the derivative of sin is cos
$\sin^{\bullet} = \lambda x.\ \cos(x)$	a true fact: it is that function that maps x to cos(x)
$\sin^{\bullet}(x) = \cos(x)$	an *abuse* of notation for one of the above (first x is *binding*)

The Rate of Change

Differential Equations for Functional Programmers (like me)

A *differential equation* means find any f such that $f^\bullet = F$ (where f may occur in F!)

$f^\bullet = \cos$	solution: $f = \sin$ (no recursion in definition of $f^\bullet$)
$f^\bullet = \lambda x.\ -k \cdot f(x)$	solutions (for any C): $f = \lambda x.\ Ce^{-k \cdot x}$
$d/dx\ f = -kf$	common abuse of the above equation (x is *unapplied* on the right! what if not pointwise???)
$f = Ce^{-kx}$	common abuse of the above solution (x is *unbound* on the right!)
$d/dx\ f(x) = \ldots$	Proper Liebniz notation, mostly useful for partial differentiation

Pointwise abuse: To make things a bit shorter, we overload arithmetic operators pointwise from numbers to number-valued functions (constants are overloaded to constant functions) so we can write:

$f^\bullet = -2f$ or $f^\bullet(x) = -2f(x)$ (meaning $f^\bullet = \lambda x.\ (\lambda y.\ -2)(x) \cdot f(x)$)

Note that omitting "(x)" does not always work, e.g. when considering "non-pointwise equations", then the differentiation parameter must be explicit:

$f^\bullet(x) = -2f(x-1)$

Fortunately we don't have many of such "history-dependant" derivatives.

The Rate of Change of Concentrations

Therefore, for example:

$$[P]^\bullet = -k[P]$$

means

$$X^\bullet = -kX$$

means

$$(\lambda t.\ X(t))^\bullet = \lambda t.\ -k\cdot(X(t))$$

with solution(s) for X:

$f_C = \lambda t.\ Ce^{-kt}$ for each initial concentration C

In general we have a system of differential equations among concentrations:

$$[A]^\bullet = \ldots [A] \ldots [B] \ldots [C] \ldots$$
$$[B]^\bullet = \ldots [A] \ldots [B] \ldots [C] \ldots$$
$$[C]^\bullet = \ldots [A] \ldots [B] \ldots [C] \ldots$$

which may be hard to solve symbolically, in which case we will have to solve it numerically (for some specific values of initial concentrations).