Reactive Systems

Luca Cardelli

University of Oxford

Applied Systems Biology Course 2023-09-19

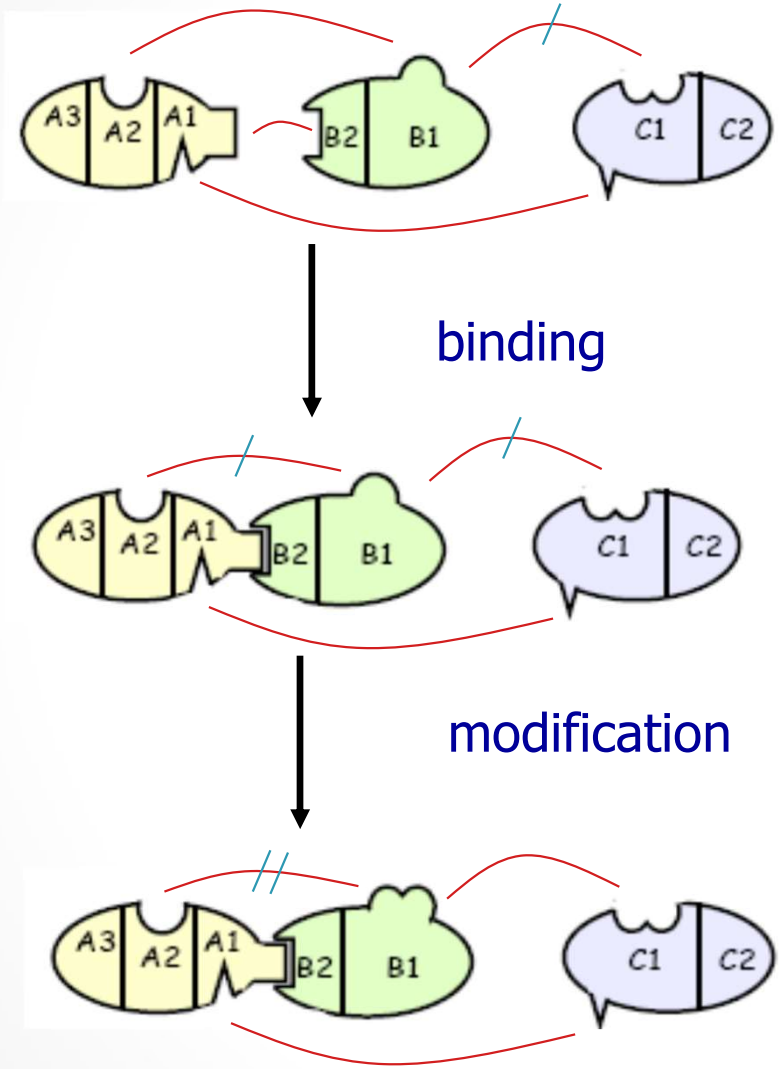
https://sites.google.com/view/applied-system-biology-course/

http://lucacardelli.name

Outline

- Reactive Systems
  - Processes vs. Functions
- Processes and Chemistry
  - Chemical modeling
  - Biochemical modeling (complexation etc.)
- Modeling Combinatorial Systems
  - General strong points of “agent-based” or “reactive” modeling languages

Dynamic Interactions



possible bindings

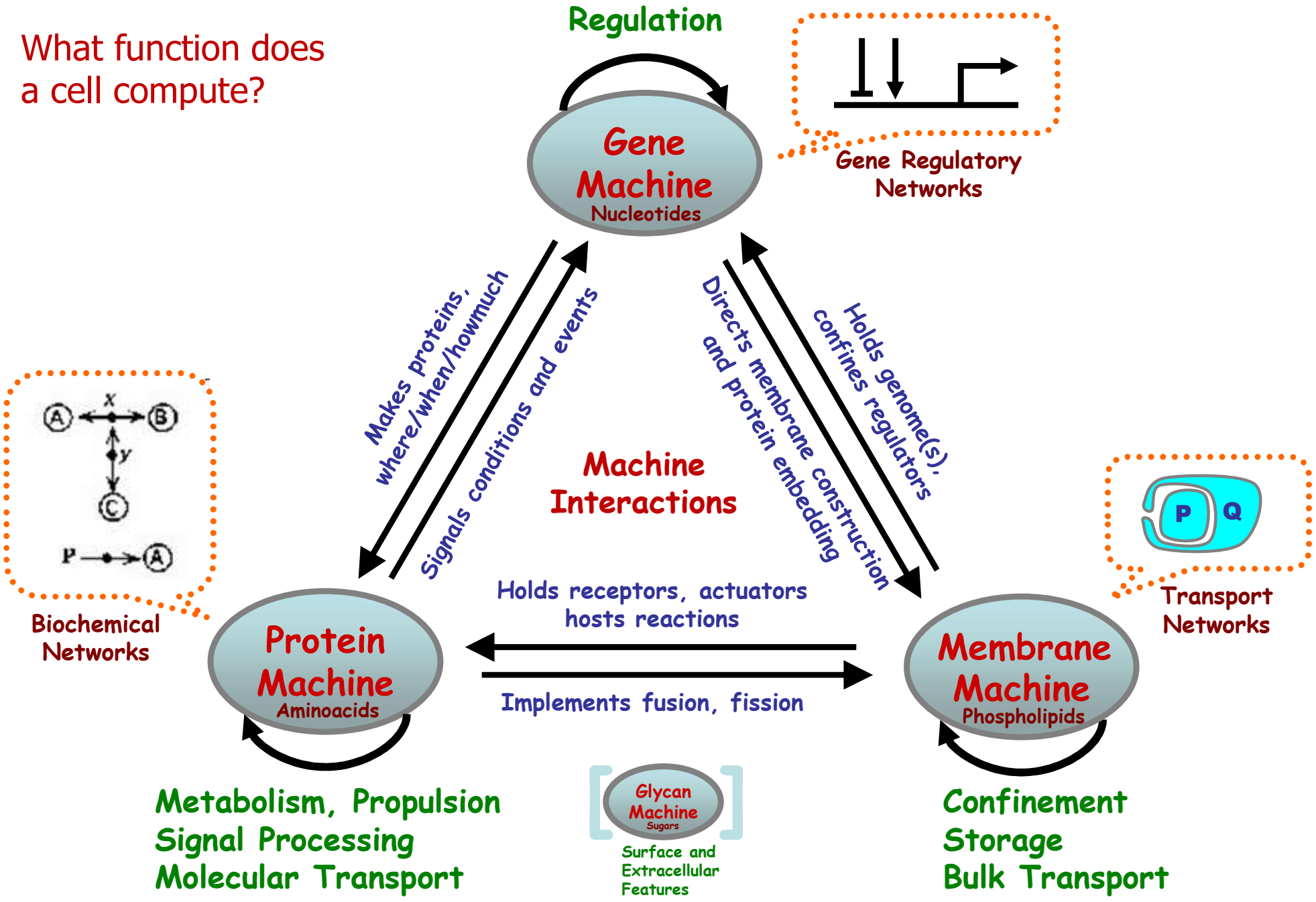
fewer bindings

different bindings

Mathematical modeling: What function does a protein compute?

Macromolecules and their Machines

What function does a cell compute?



Reactive Systems

- Developed in Computer Science to get away from the strict notion of input/output computation
- Even more fundamentally, to get away from the very notion of *functions* as mathematical modeling tools

- We realized in the 60's-80s' that we could not *practically* model/implement operating systems and computer networks as imperative programs, because of combinatorial state explosion (which is also pervasive in biology)
- And we struggled *theoretically* to model them via appropriate mathematical functions, because of concurrency and nondeterminism (which are also pervasive in biology)
- What function does the internet (or a protein / gene / membrane / cell) compute?

- This led to a fundamental change in point of view of how to model systems in general

How does X compute

- Change the question from *What* to *How*
  - How does an Operating System compute?
  - How does the Internet compute?
  - How does a Cell compute?

- I.e.: what *steps* does X perform?
  - We can talk about what *steps* a subsystem performs in *reaction* to a *stimulus* from the environment
  - And what are the local consequences of those steps
  - (And this turns out to include functions as a special case)

What is reactive computation?

- Functions
  - Functions are supposed to *terminate* and give a final answer
  - But if e.g. the internet terminates, it's useless: whatever answers it gives, they are never final. And if a cell terminates, it's dead.

- Reactive Systems (by many names: *agent-based,* etc.)
  - Systems that accept inputs and provide outputs, but not necessarily in a "functional" way, e.g. because they have multiple interaction points, they have internal state, or have "a mind of their own“ that does not even depend on the inputs.
  - They *react* to the environment.
    The environment *reacts* to them.
  - In this view computation is *reaction*, or more symmetrically is *interaction*, or is *communication* when something is exchanged during interaction.

Functions vs. Processes[1]

We analyze the notion of "computation step" for functions, and we generalize it to processes

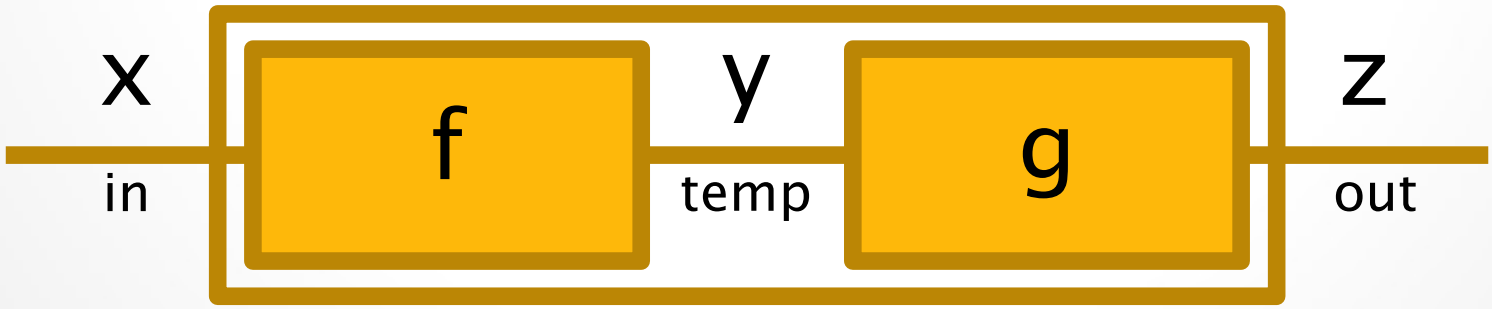
(1) A "Process" is a component of a reactive system

Function Diagrams

- We use diagrams to emphasized simple computation steps

- (A function of two inputs would be a function of a single input that is a pair)

- Function composition (g o f)
  N.B. we rename *out* to *temp* in f, and *in* to *temp* in g,
  so they connect through their now common channel *temp*



Process Diagrams

- Multiple input and multiple output *channels*

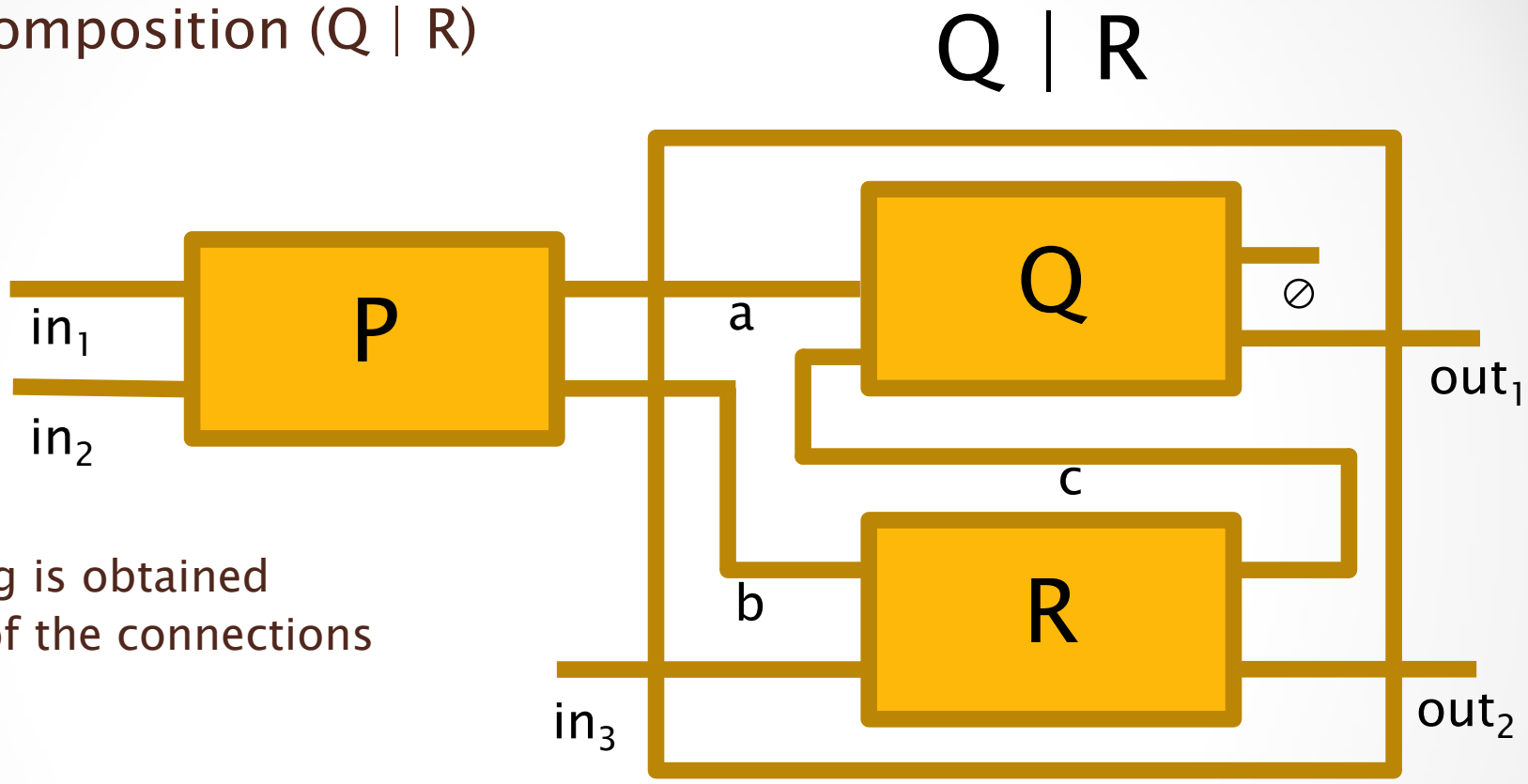
This is not a function of 2 inputs. Don't think And-gate (a Boolean function with 2 in 1 out)

Think Flip-Flop (a "thing" with 2 independent set/reset inputs and 2 separate outputs)

- In “pure interaction” there is actually no difference between input and output channels: they just interact symmetrically with the environment
- But in “communication” there is a direction in which messages a received or sent. We may or may not label receive-channels as “in” and send-channels as “out”
- Specific “channel names” become fundamental to describe how processes interconnect (unlike functions, which have 1 default input and 1 default output channel name)

Process Diagrams

- Process composition (Q | R)



A specific wiring is obtained
by (re)naming of the connections
ahead of time

Rules: P | 0 = P
P | Q = Q | P
P | (Q | R) = (P | Q) | R

0 is the null process that has no channels

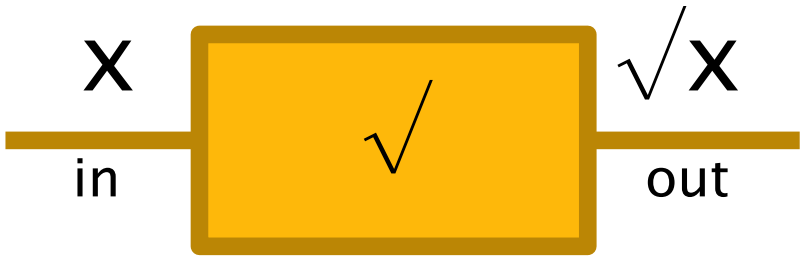
(P | P ≠ P in general)

Functions as Processes

function *input* *apply*

$$f(x) = \sqrt{x}$$ — *output*

Reactive Diagram



"read from in into x; then write $\sqrt{x}$ to out"

Reactive Syntax

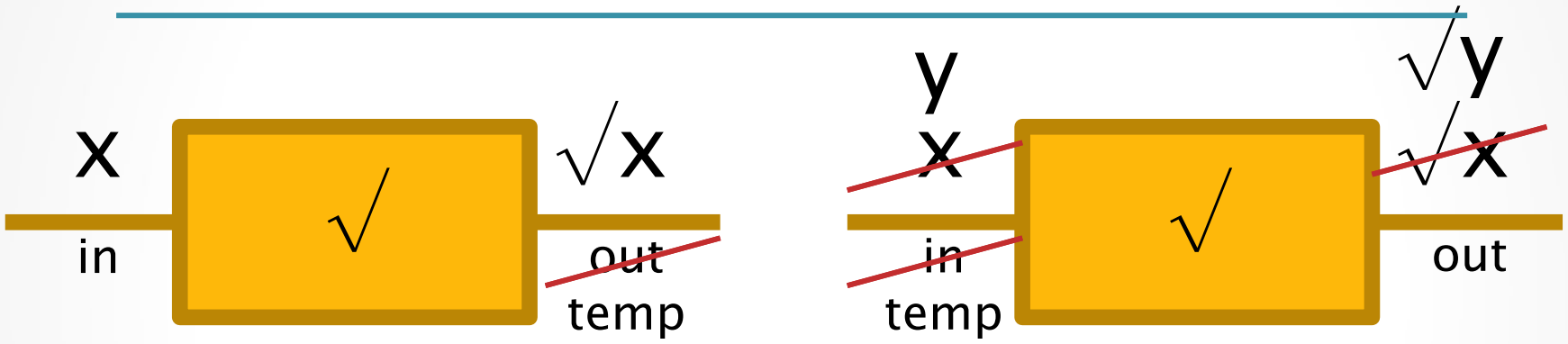
read *write* *channels*

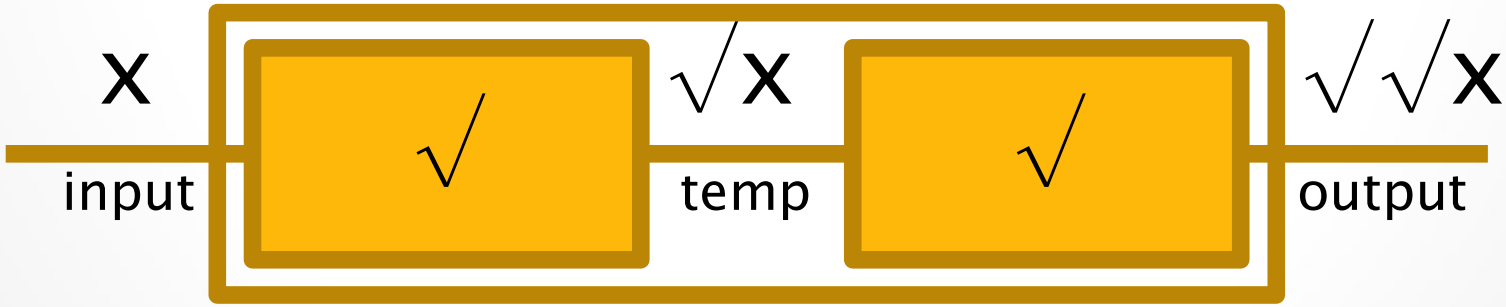
$$f = ?in(x);\ !out(\sqrt{x})$$

input *output*

Composing Functions

$$g(x) = (f \circ f)(x) \quad (= f(f(x)))$$

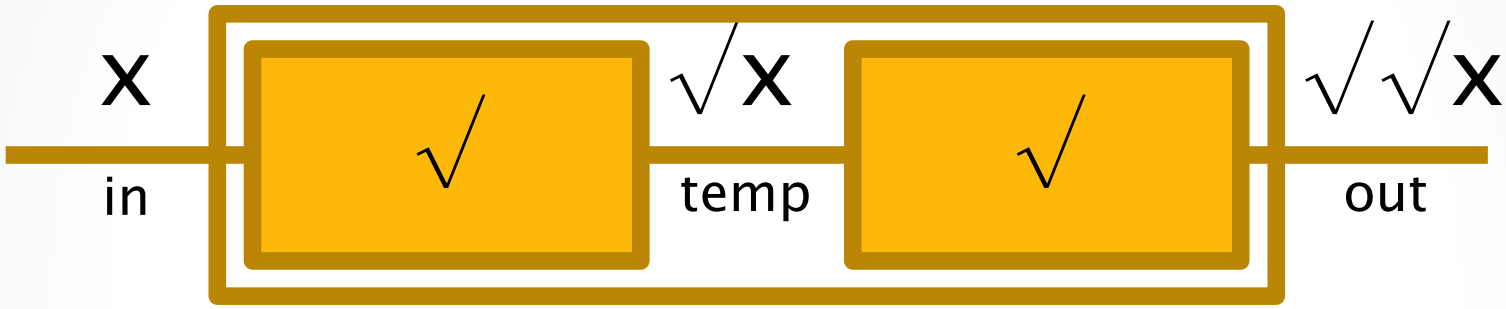




Composing Functions

$$g(x) = (f \circ f)(x)$$

Reactive Diagram



"create a *new* channel and use it to compose two copies of f"

Reactive Syntax

channel creation

private channel

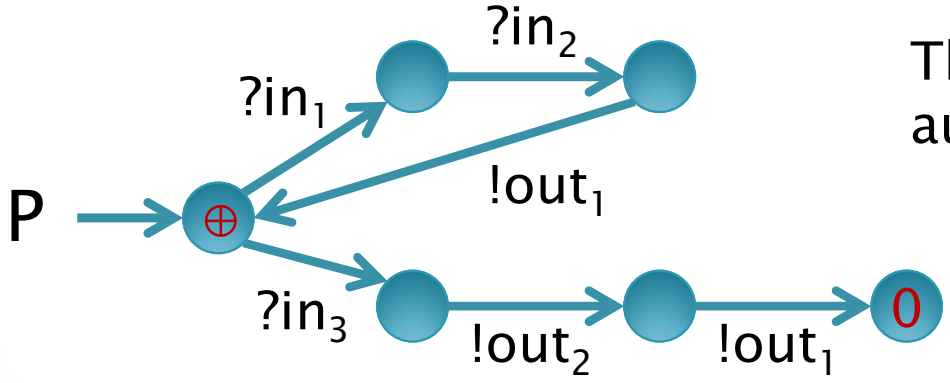
composition

$$g = (\nu\ \text{temp})$$
$$?in(x);\ !temp(\sqrt{x})\ |$$
$$?temp(y);\ !out(\sqrt{y})$$

A Process that is not a Function

Reactive Diagram

Reactive Skeleton



The 'skeleton' automaton

Reactive Syntax

$$P = ?in_1(x);\ ?in_2(y);\ !out_1(x+y);\ P \oplus ?in_3(z);\ !out_2(\sqrt{z});\ !out_1(2z);\ 0$$

choice

recursio

Ex. Vending Machine

A candy for 2 coins

VM =

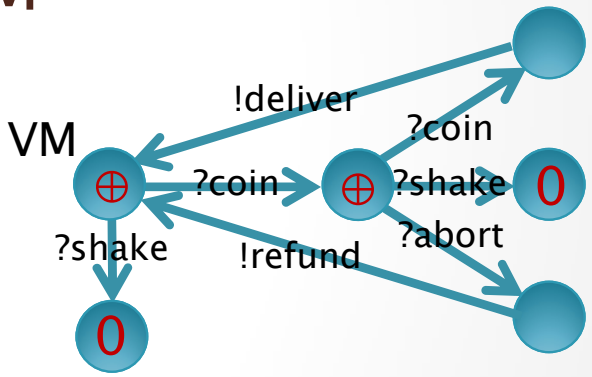
?coin(c1);

(?coin(c2); !deliver(candy); VM

⊕ ?abort(); !refund(c1); VM

⊕ ?shake(); 0)

⊕ ?shake(); 0



Note that we are not describing the environment.

Dynamic Networks

- Here is where (static) diagrams fail, and syntax *has* to take over
  - Proliferation/degradation

$$P = \ ?in(); (P \mid P)$$

$$P = \ ?in(); 0$$

  - Interface modification

$$P = \ ?in(x); !x(); Q$$

channel passing!

  - Biological (and software) networks do both things a lot, as opposed to e.g. electrical networks

That's π-calculus

- To compose processes P we need:
  - Composition: P | P (with identity elem. 0)
  - Channel cration: $(\nu\ x)\ P$ (with x bound in P)
  - Recursion: *P (equal to P | *P)
- To perform computation steps we need:
  - Channel reading: ?c(x); P (with x bound in P)
  - Channel writing: !c(M); P (with message M)
  - Choice: $P \oplus P$ (with identity elem. 0)
- … and channels can be sent as messages!

Generalizing Functions and Automata

- Unlike functions…
  - Processes have multiple, explicitly named, input and output channels.
  - Processes can run in *parallel*, can *deadlock* on their inputs, and can be *nondeterministic* in their outputs.

- Unlike automata (FSA)…
  - Processes can transmit data (not just change state).
  - While automata ‘talk’ to input strings, processes ‘talk’ to other processes: processes are communicating automata.
  - Processes are not “finite state”; they can express unbounded computation in time (divergence) and space (proliferation).
  - They have dynamic connectivity: networks can reshape themselves.

The Kinetics[1] of Computation

1) how things *move*, in this case what steps they take and possibly at what rate.

Function Kinetics

- Functions have a single kinetic law:
  - *Application rule*:

    computes uniquely to

    If $f(x) =_{def} M\{x\}$ then $f(a) \rightarrow M\{a/x\}$

    e.g.; $f(x) =_{def} not(x)$ then

    $f(true) \rightarrow not(x)\{true/x\} = not(true) .... \rightarrow false$

- No other kinetic rule is strictly required
  - E.g., no arithmetic: “variable shuffling” is enough to compute anything [Church/Turing]

- The application rule is *how* functions compute

Process Kinetics

- Processes have one kinetic law:
  - *Communication rule*

may compute to

$$(?c(x);P\{x\}) \oplus P' \mid (!c(a);Q) \oplus Q' \rightarrow P\{a/x\} \mid Q$$

- plus one important identity:
  - *Extrusion rule*

is the same as (not a computation step)

$$((\nu\ x)P) \mid Q = (\nu\ x)(P|Q) \quad \text{for } x \text{ not occurring in } Q$$

- No other kinetic rule is strictly required
  - "channel shuffling" is enough to compute anything [Milner]
- The communication rule is *how* reactive systems compute

Ex. VM in environment

VM = ?coin(c1);
(?coin(c2); !deliver(candy); VM
⊕ ?abort(); !refund(c1); VM
⊕ ?shake(); 0)
⊕ ?shake(); 0

BUYER =
!coin(10p);
!coin(10p);
?deliver(yum);
EAT

VM | BUYER

→ ?coin(c2); !deliver(candy); VM | !coin(10p); ?deliver(yum); EAT
⊕ ?abort(); !refund(10p); VM
⊕ ?shake(); 0

→ !deliver(candy); VM | ?deliver(yum); EAT

→ VM | EAT

There can be a rate associated with each channel, determining the speed of interactions.

Equivalence

- The *"what does X compute?"* question can be replaced by equivalence: two systems compute the same thing (whatever that is) if they can replace each other in *every possible situation*, according to the kinetic laws.
- Two functions are equivalent if they can replace each other in *every context* made up by other functions *(Extensionality)*
  - If they produce the same outputs from the same inputs.
  - The Church-Rosser theorem states that the kinetic law for functions leads to deterministic answers: this is non-trivial.
- Two processes are equivalent if they can replace each other in *every environment* made up by other processes *(Contextual congruence)*
  - A proof technique for process equivalence is based on showing *bisimilarity*: for each step one process can make, the other can choose to make a similar step ending up again in equivalent processes.
  - One of the main theorems states that bisimilarity is a congruence, i.e., that if two systems are bisimilar, then they can be exchanged for one another in every environment with no observable difference.

Chemical Networks as Reactive Systems

Chemical Systems

Reactions:

$$A_1 + A_2 \rightarrow^r B_1 + B_2$$

Deterministic reaction kinetics

$$d[A_i]/dt = -r[A_1][A_2]$$ Mass Action Law

(assuming $B_i \neq A_j$ for all i,j)

Stochastic reaction kinetics

Chemical Master Equation -> CTMC

As Reactive Systems

Uniquely-named *asymmetric* ($A_1 \neq A_2$) reaction c:

$$c:\ A_1 + A_2 \rightarrow^r B_1 + B_2$$

A reaction is *ambiguous* about how the lhs species transfer to the rhs

Processes:

channel with rate

$A_1 = ?c_r;\ B_1$

$A_2 = !c_r;\ B_2$

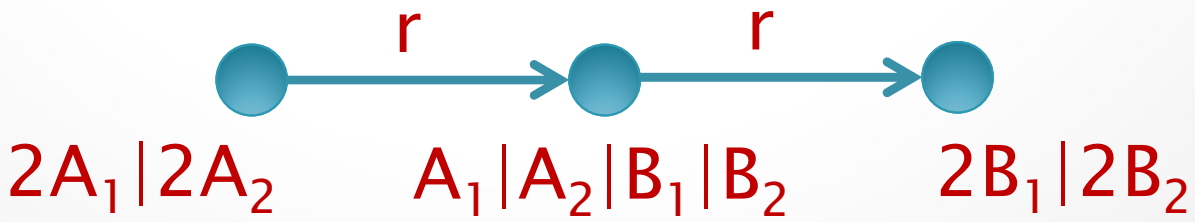
(the name of the reaction becomes the channel)

The reactive system must resolve the ambiguity

or: $A_1 = ?c_r;\ B_2$, $A_2 = !c_r;\ B_1$

or: $A_1 = ?c_r;\ (B_1|B_2)$, $A_2 = !c_r;\ 0$

The (quantitative) kinetic laws for processes lead to a CTMC semantics. E.g., with initial conditions $2A_1|2A_2$, the CTMC is:



As Reactive Systems

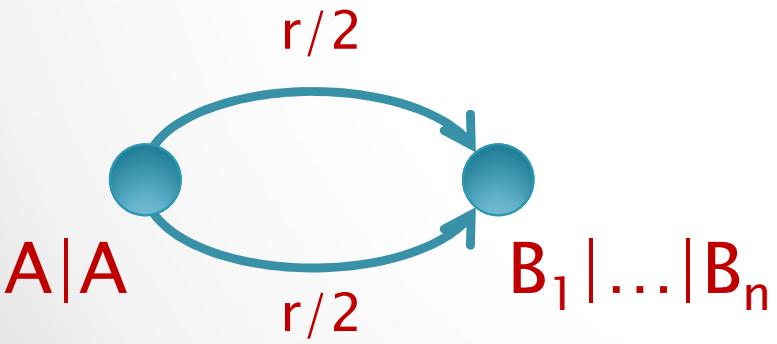
Uniquely-named *symmetric* reaction c:

$$c:\ A + A \ \rightarrow^{r}\ B_1 + \ldots + B_n$$

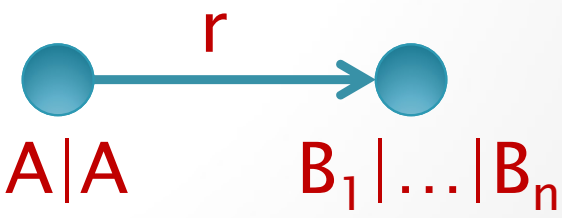
Discrete reaction kinetics (A must interact with a copy of itself):

$$A = ?c_{r/2};\ (B_1|\ldots|B_i) \oplus !c_{r/2};\ (B_{i+1}|\ldots|B_n) \qquad 0 \le i \le n$$

With initial conditions A|A (two molecules), the CTMC is as follows; note that each copy of A can do an input or an output, so there are two possible paths to the outcome:



That is:



From Reactions to Processes

v_1: $A+B \rightarrow_{k_1} C+C$

v_2: $A+C \rightarrow_{k_2} D$

v_3: $C \rightarrow_{k_3} E+F$

v_4: $F+F \rightarrow_{k_4} B$

Interaction Matrix

channels (1 per reaction)

Half-rate for symmetric reactions

processes (1 per species)

	$v_{1(k1)}$	$v_{2(k2)}$	$v_{3(k3)}$	$v_{4(k4/2)}$
A	?;(C|C)	?;D		
B	!;0			
C		!;0	τ;(E|F)	
D				
E				
F				?;B !;0

Fill the matrix by columns:

Degradation reaction v_i: $X \rightarrow^{ki} P_i$
add $\tau;P_i$ to $\langle X,v_i \rangle$.

Asymmetric reaction v_i: $X+Y \rightarrow^{ki} P_i$
add $?;P_i$ to $\langle X,v_i \rangle$ and $!;0$ to $\langle Y,v_i \rangle$

Symmetric reaction v_i: $X+X \rightarrow^{ki} P_i$
add $?;P_i$ and $!;0$ to $\langle X,v_i \rangle$

Read out the processes by rows:

$A = ?v_{1(k1)};(C|C) \oplus ?v_{2(k2)};D$

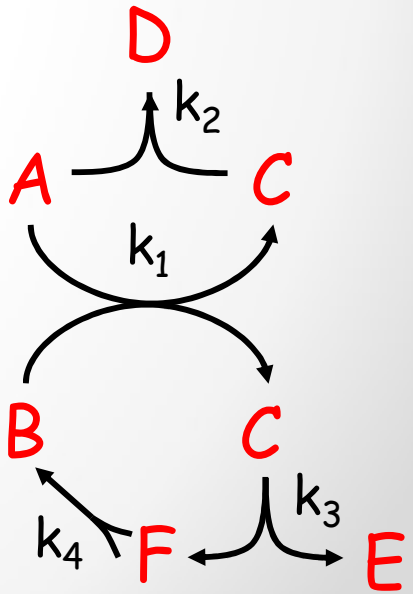
$B = !v_{1(k1)};0$

$C = !v_{2(k2)};0 \oplus \tau_{k3};(E|F)$

$D = 0$

$E = 0$

$F = ?v_{4(k4/2)};B \oplus !v_{4(k4/2)};0$



Different Processes for the same Reactions

- That was a *systematic* way to translate reactions to processes, akin to the way the stoichiometric matrix leads systematically to generating mass action ODEs.
- There can be multiple reaction systems that produce the same ODEs: reactions are *finer* than ODEs.
- Here there can be multiple process systems that produce the same reactions: processes are *finer* than reactions.
- Hence there can be *better or worse* ways to get processes that match certain reactions.
- That is, different processes that produce *more compact and/or modular models*, but with the *same kinetics*.

Ex: Catalysis

- Two reactions, same catalyst C
  - According to the general scheme the catalyst uses one channel for each reaction it catalyzes

    a: $A + C \to^{r} C + B$

    b: $D + C \to^{r} C + E$

    $C = !a_r; C \oplus !b_r; C$

    $A = ?a_r; B$

    $D = ?b_r; E$
  - Modularizing: the catalyst has its own catalysis channel c, used for all the reactions it catalyzes:

    $A + C \to^{r} C + B$

    $D + C \to^{r} C + E$

    $C = !c_r; C$

    $A = ?c_r; B$

    $D = ?c_r; E$
  - A very minor improvement in model size here, but these improvement compound in larger models

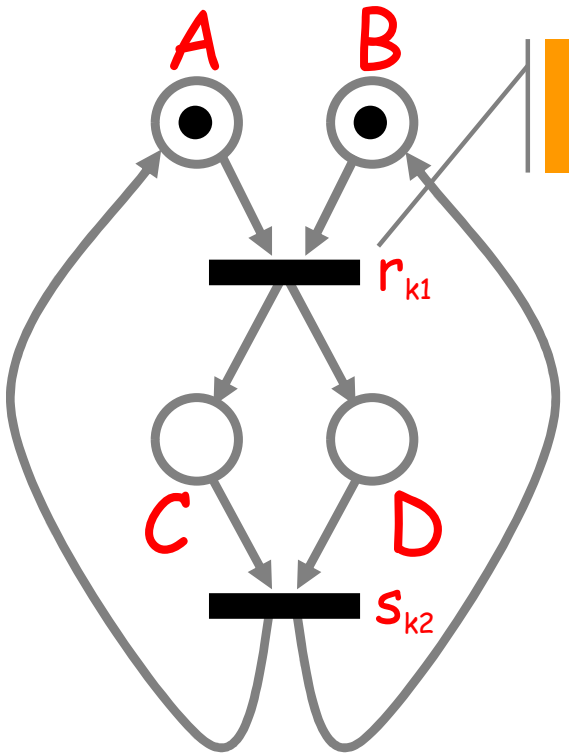
Chemistry vs. π-calculus

Chemical reactions

$r: A + B \rightarrow_{k1} C + D$

$s: C + D \rightarrow_{k2} A + B$

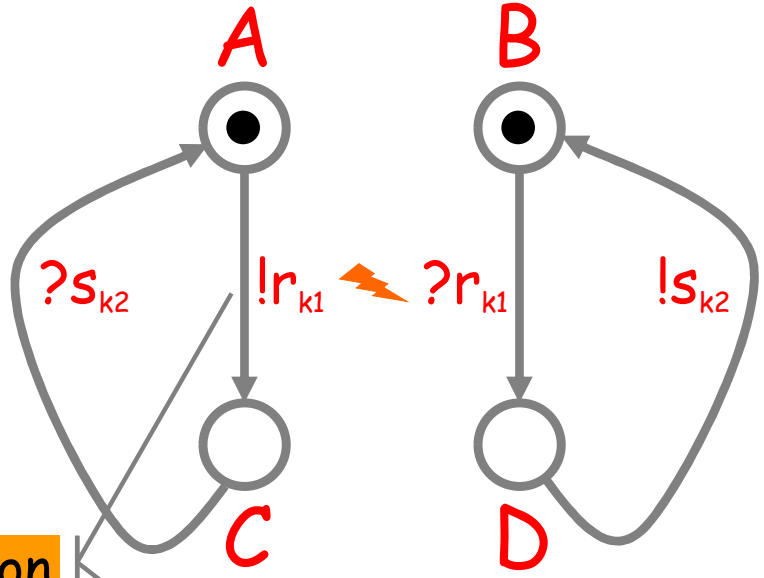
Does A become C or D?



Reaction oriented

1 line per reaction

Reactive system (π)



Interaction oriented

1 line per component

$A = !r_{k1}; C$

$C = ?s_{k2}; A$

$B = ?r_{k1}; D$

$D = !s_{k2}; B$

A becomes C not D!

The same "model"

Maps to a CTMC	Maps to a CTMC

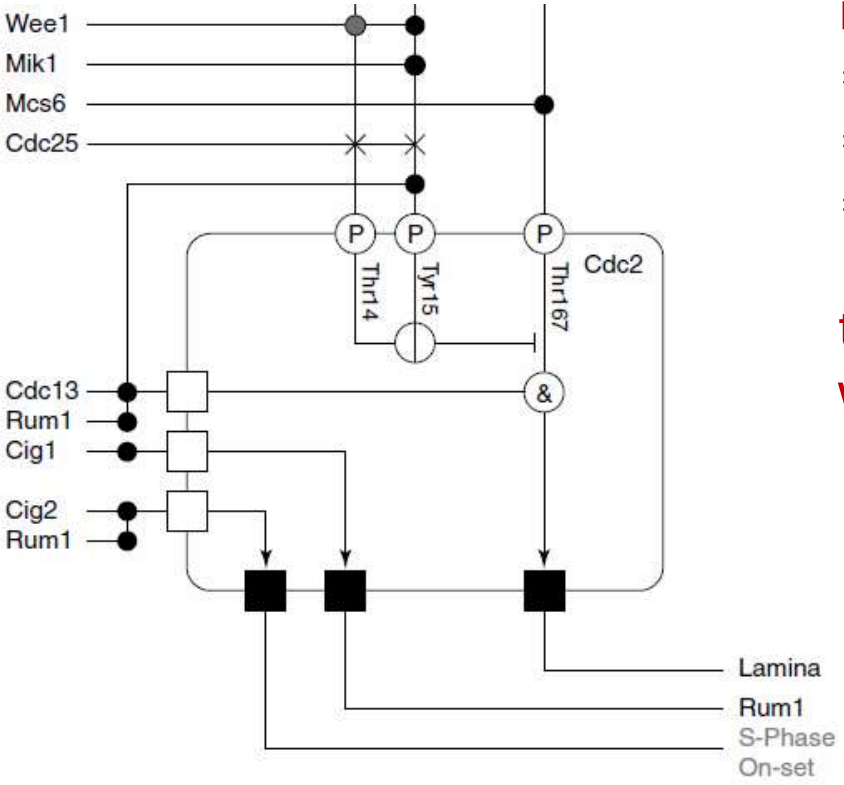
A Petri-Net-like representation. Precise and dynamic, but not modular, scalable, or maintainable.

A compositional graphical representation (precise, dynamic *and* modular) and the corresponding calculus.

Biochemical Networks as Reactive Systems

Molecules with State

- Explosion of species, reactions, and state space.



(b) Proposed improvements of graphical representation of fission yeast Cdc2

n modification sites
= 2^n molecular states
= 2^n 'species'
= 2^n ODEs (mass action)

the master equation will have 2^n ODEs for *each* molecule!

BIOSILICO Vol. 1, No. 5 November 2003

A graphical notation for biochemical networks

Hiroaki Kitano

Connected Molecules

- Further combinatorial explosion

n states -- m states = nxm states

$2^{n1} \times 2^{n2} \times \ldots \times 2^{nm}$ = BIG

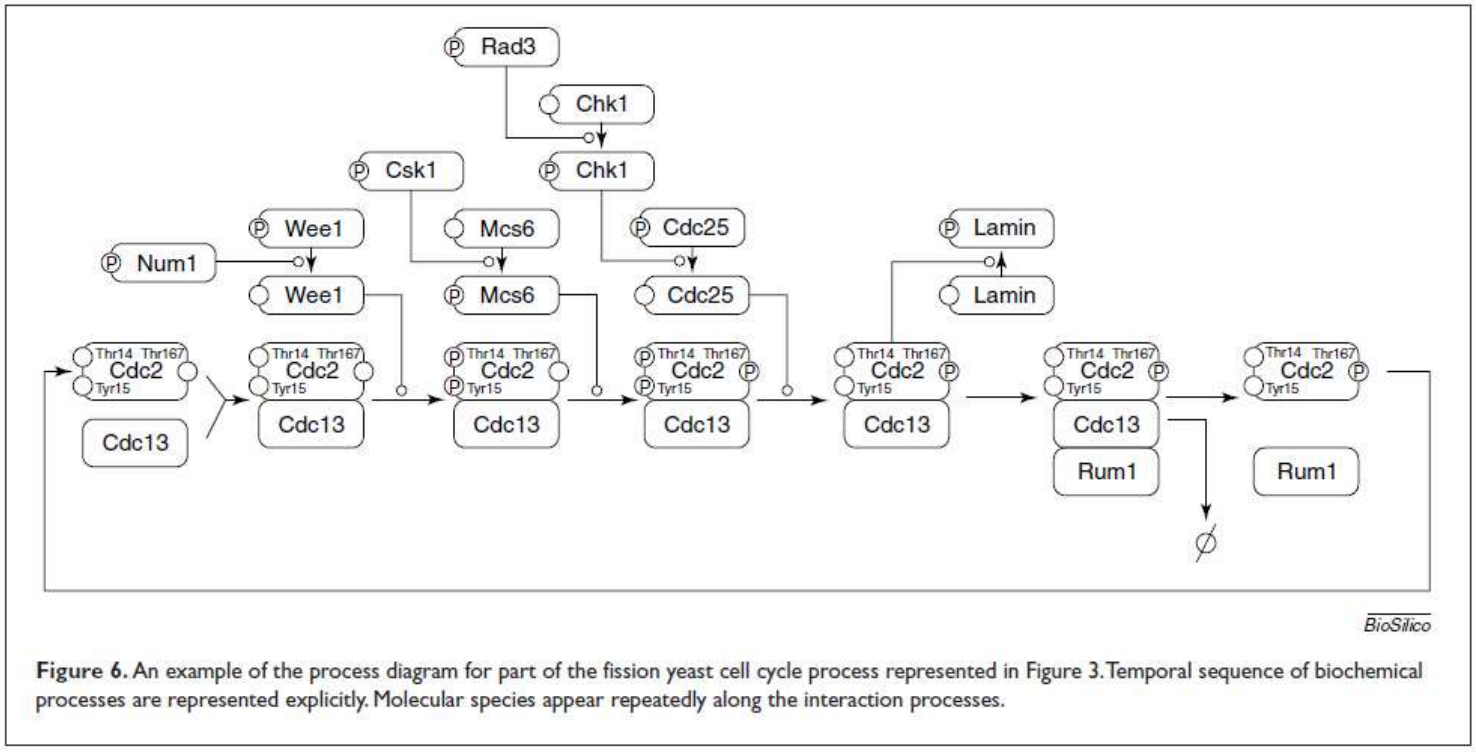


Figure 6. An example of the process diagram for part of the fission yeast cell cycle process represented in Figure 3. Temporal sequence of biochemical processes are represented explicitly. Molecular species appear repeatedly along the interaction processes.

BIOSILICO Vol. 1, No. 5 November 2003

A graphical notation for biochemical networks

Hiroaki Kitano

Iterated Connections (Polymers)

- ‘Infinite’ explosion

An *actually infinite* number of species and ODEs

p_1 (polymer of length 1)
p_2 (polymer of length 2)
p_3 (polymer of length 3)
…

Copolymer equation [edit]

An alternating copolymer has the formula: -A-B-A-B-A-B-A-B-A-B-, or $-(-A-B-)_n-$. The molar ratios of the monomer in the polymer is close to one, which happens when the reactivity ratios r_1 & r_2 are close to zero, as given by the Mayo-Lewis equation also called the **copolymerization equation**:[11]

$$\frac{d[M_1]}{d[M_2]} = \frac{[M_1]([M_1] + r_1[M_2])}{[M_2]([M_1] + r_2[M_2])}$$

where $r_1 = k_{11}/k_{12}$ & $r_2 = k_{22}/k_{21}$

WIKIPEDIA
The Free Encyclopedia

π-calculus for Biochemistry

- Biochemistry here means
  - *Direct* modeling of complexation and polymerization, which are fundamental biochemical features.
  - That is, a complex is not a "new species": it is a structure formed by existing basic species, which can also break apart.

- We now need the *full* π-calculus
  - We need to create new channels to represent new *complexation bonds*.
  - We need value-passing so the components of a complex can operate on those bonds: we need to pass *channels over channels*.

Complexation

$$A + B \quad {}^{s}\!\leftrightarrow^{r} \quad AB$$

There is no good notation for this reaction in chemistry: AB is considered as a separate species (which leads to combinatorial explosion of models).

But there is a way to write this precisely in π-calculus. Let there be a single public *association* channel a_r at rate r, and many private *dissociations* channels d_s at rate s, one for each complexation event (these are dynamically created by the new-channel operator ν):

$$A_{free} = (\nu\ d_s)\ !a_r(d_s);\ A_{bound}(d_s)$$
$$A_{bound}(d_s) = !d_s;\ A_{free}$$

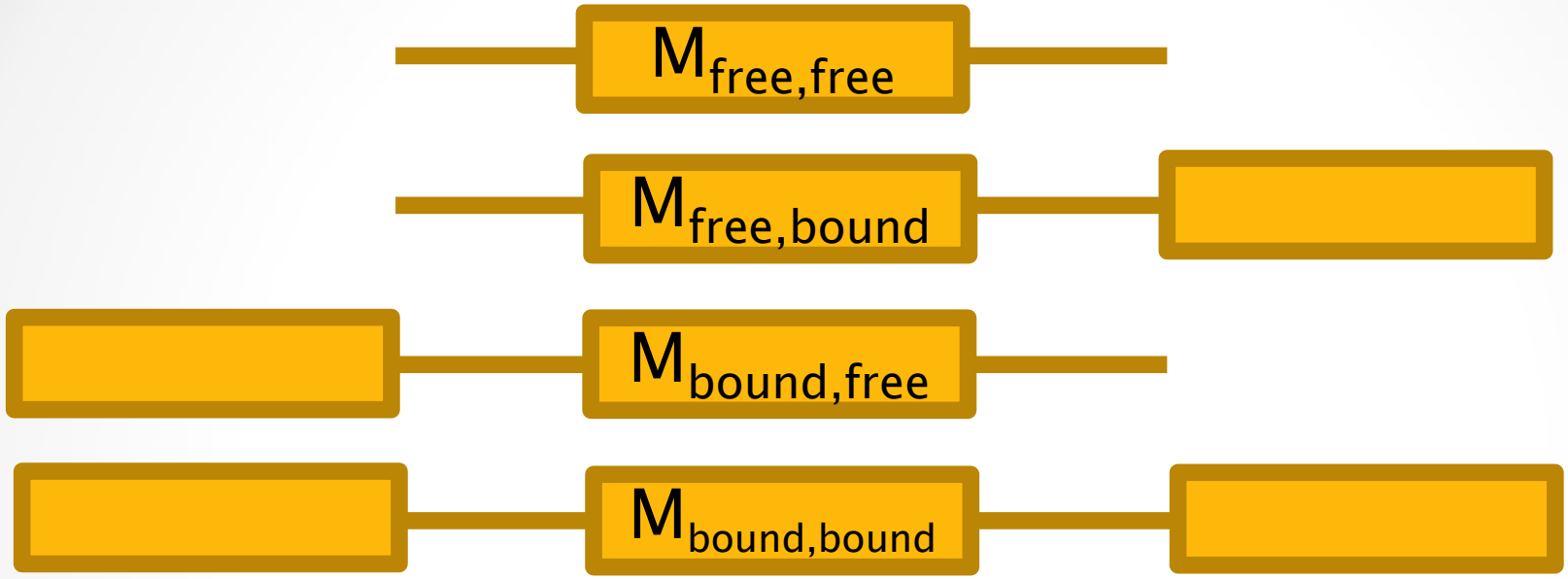
$$B_{free} = ?a_r(d_s);\ B_{bound}(d_s)$$
$$B_{bound}(d_s) = ?d_s;\ B_{free}$$

Note that we are describing A *independently* of B: as in the catalysis example, A could form complexes with many different species over the a_r channel.

More compactly:

$$A = (\nu\ d_s)\ !a_r(d_s);\ !d_s;\ A$$
$$B = ?a_r(d_s);\ ?d_s;\ B$$

Polymerization



$$M_{free,free} = ?a(b);\ M_{bound,free}(b) \oplus (\nu\ b)\ !a(b);\ M_{free,bound}(b)$$
$$M_{bound,free}(l) = !l;\ M_{free,free} \oplus (\nu\ b)\ !a(b);\ M_{bound,bound}(l,b)$$
$$M_{free,bound}(r) = !r;\ M_{free,free} \oplus ?a(b);\ M_{bound,bound}(b,r)$$
$$M_{bound,bound}(l,r) = !l;\ M_{free,bound}(r) \oplus !r;\ M_{bound,free}(l)$$

$$M_{free,free} \mid M_{free,free} \rightarrow (\nu\ b)\ M_{bound,free}(b) \mid M_{free,bound}(b)$$

Polymerization

- Polymerization is iterated complexation
  - It can be represente in π-calculus *finitely*, with one process (definition) for each monomer state.
  - Note that polymerization cannot be described *finitely* in chemistry (or ODEs) because there it needs one reaction for each *length* of polymer.
  - The reason it works in π-calculus is because of the ν operator. It enables the finite representation of systems of potentially unbounded complexity.
  - As in real biochemistry, where the structure of each monomer is coded in a finite piece of DNA, and yet unbounded-length polymers happen.

Simulation

- It is possible to run simulations (particularly Gillespie-like stochastic simulation)
- For reactive systems with very large or even infinite numbers of species and reactions (like polymerization or unbounded complex formation) where it would be impossible to even write them all down.
- Without ever computing all the reactions or all the possible complexes
- By producing them only during the simulation, as the need arises
- Without any kind of size cut-off or approximation.

Biochemistry vs. π-calculus

n — domains: A, B, C

2n — domain reactions:
$A \leftrightarrow A_p$
$B \leftrightarrow B_p$
$C \leftrightarrow C_p$

1 — complex: ABC

2^n — species:
ABC
A_pBC
AB_pC
ABC_p
A_pB_pC
A_pBC_p
AB_pC_p
$A_pB_pC_p$

$2n(2^{n-1})$ — reactions (twice number of edges in n-dim hypercube):
$ABC \leftrightarrow A_pBC$
$ABC \leftrightarrow AB_pC$
$ABC \leftrightarrow ABC_p$
$A_pBC \leftrightarrow A_pB_pC$
$A_pBC \leftrightarrow A_pBC_p$
$AB_pC \leftrightarrow A_pB_pC$
$AB_pC \leftrightarrow AB_pC_p$
$ABC_p \leftrightarrow A_pBC_p$
$ABC_p \leftrightarrow AB_pC_p$
$A_pB_pC \leftrightarrow A_pB_pC_p$
$A_pBC_p \leftrightarrow A_pB_pC_p$
$AB_pC_p \leftrightarrow A_pB_pC_p$

Stoichiometric Matrix (species x reaction),

$2^n \times 2n(2^{n-1})$

The matrix is very sparse, so the corresponding ODE system is not dense. But it still has 2^n equations, one per species, plus conservation equations ($[ABC]+[A_pBC]$=constant, etc.).

System description is exponential in the number of basic components.

2n — domain reactions:
$A \leftrightarrow A_p$
$B \leftrightarrow B_p$
$C \leftrightarrow C_p$

processes

2n:
$A = ?kn;A_p$ $\quad A_p = ?ph;A$
$B = ?kn;B_p$ $\quad B_p = ?ph;B$
$C = ?kn;C_p$ $\quad C_p = ?ph;C$

n: $A \mid B \mid C$

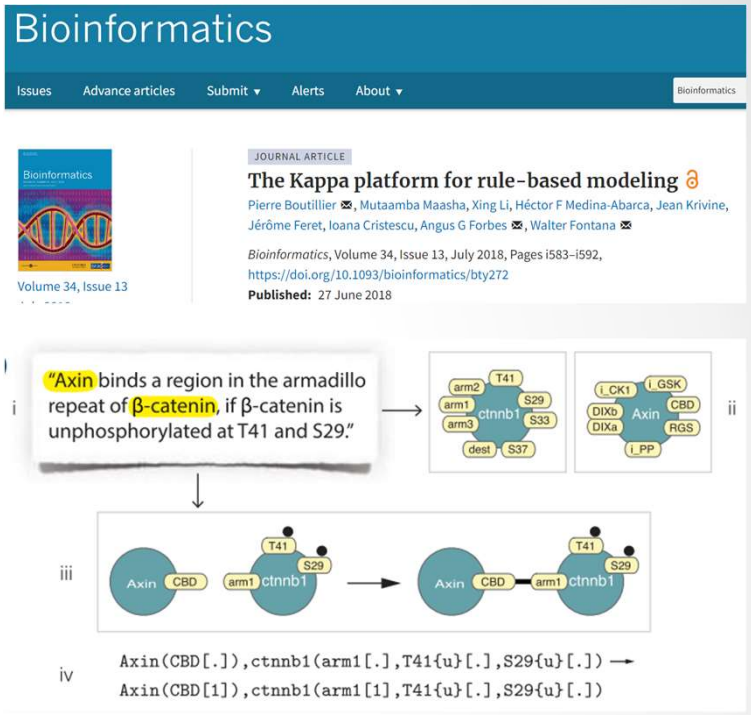
System description is linear in the number of basic components.

(Its "run-time" behavior or analysis potentially blows-up just as in the previous case, but its description does not.)

Applications

Protein Machine

- Kappa is a (much) more convenient reactive system modeling framework for biochemical systems
- It takes complexation and post-translational modification as primitives, instead of encoding them by π-calculus "plumbing" with channels-over-channels. In diagram form, it looks very much like this, but with its own peculiar syntax:
- Its kinetic laws are however more complex. The original paper on Kappa [Danos, Laneve] gives a non-trivial translation from Kappa to π-calculus.
- Kappa has since been used to model compactly, simulate, and analyze biochemical systems of huge combinatorial complexity [Fontana et al.] including stochastic kinetics.
- Its level of "rule-based" modeling approach is very close to biochemical statements such as: *"protein A is phosphorylated on site 1 and then binds to protein B on site 2"*



Gene Machine

Simulation of DNA replication fork dynamics and positional occupation over the DNA sequence, compared to nanopore data.

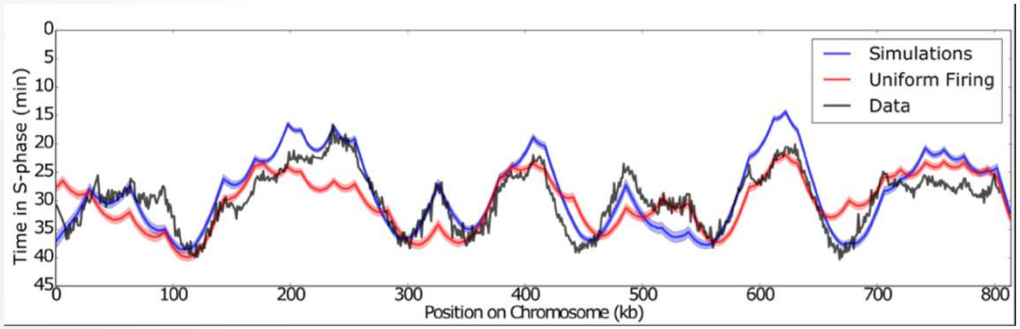
PLOS COMPUTATIONAL BIOLOGY

OPEN ACCESS PEER-REVIEWED

RESEARCH ARTICLE

The Beacon Calculus: A formal method for the flexible and concise modelling of biological systems

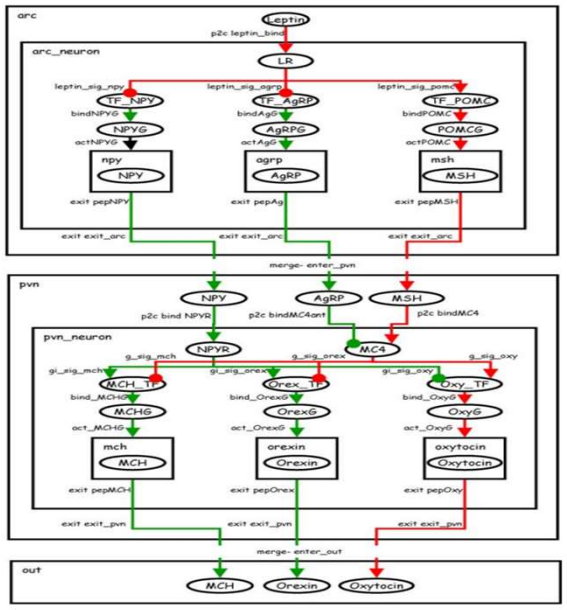
Michael A. Boemo, Luca Cardelli, Conrad A. Nieduszynski



Membrane Machine

Static Compartments

Fragment of the hypothalamic system for body weight regulation handling molecular events (receptors, signaling pathways and gene expression) within a heterogeneous cell population sequestered to distinct anatomical compartments.



ELSEVIER

Theoretical Computer Science

Volume 325, Issue 1, 28 September 2004, Pages 141-167

BioAmbients: an abstraction for biological compartments

Aviv Regev [a], Ekaterina M. Panina [b], William Silverman [c], Luca Cardelli [d], Ehud Shapiro [c]

Dynamic Compartments

Qualitative model of a whole process of virus infection and reproduction, including membrane topology transitions.

International Conference on Computational Methods in Systems Biology

↳ CMSB 2004: **Computational Methods in Systems Biology** pp 257–278 | Cite as

Home › Computational Methods in Systems Biology › Conference paper

Brane Calculi

Interactions of Biological Membranes

Luca Cardelli

Conference paper

605 Accesses | **147** Citations

Part of the Lecture Notes in Computer Science book series (LNBI,volume 3082)

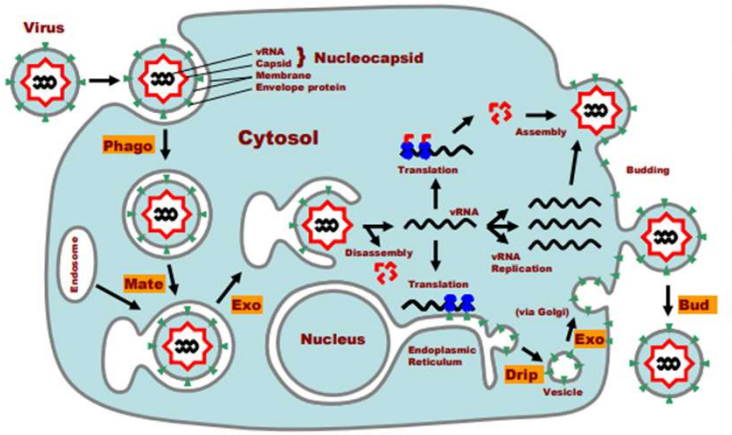


Figure 6 Viral Infection and Reproduction ([1] p.279)

Conclusions

Reactive Systems

- A model of computation
  - Many flavors, but sharing common principles and techniques
  - Suitable for nondeterministic concurrent systems with unbounded execution, unbounded proliferation, and dynamical connectivity
  - π-calculus as a minimal notation for general reactive systems
- A solution to combinatorial explosion
  - Models are exponentially (for phosphorylation/ complexation) or infinitely (for polymerization) more compact.
  - The state space is explored incrementally, and even if the state space is actually infinite (as with polymers) we can still expand it on demand and simulate it with standard techniques.
- Further Reading
  - R. Milner: **Communicating and Mobile Systems: The Pi Calculus**
  - A. Regev, E. Shapiro. **Cellular Abstractions: Cells as Computation.** NATURE vol 419, 2002-09-26, 343.
  - L. Cardelli: **From Processes to ODEs by Chemistry.** TCS 273, 261-281, 2008,
  - V. Danos, C. Laneve. **Formal molecular biology.** TCS 325(1), 69-110, 2004.
  - Pierre Boutillier et al. **The Kappa platform for rule-based modeling.** Bioinformatics. 34(13): i583-i592, 2018.