Molecules as Automata

Luca Cardelli

Microsoft Research

Emergence In Chemical Systems 2.0
Anchorage, 2009-06-25

http://LucaCardelli.name

Motivation: Cells Compute

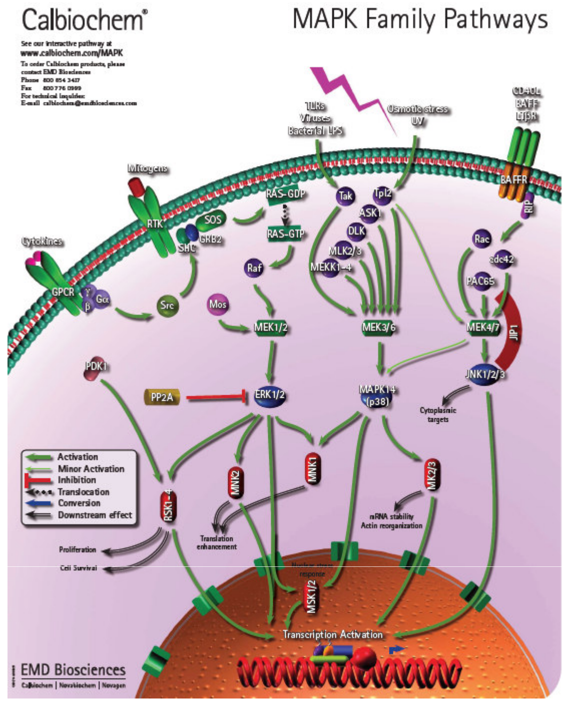
- No survival without computation!
  - Finding food
  - Avoiding predators

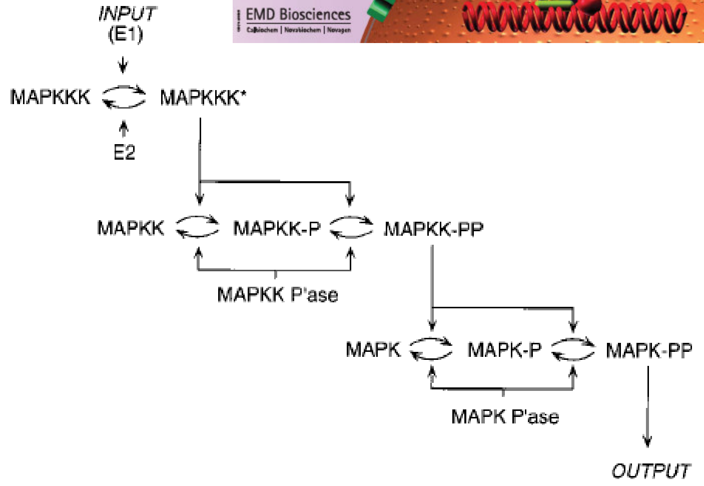
- *How* do they compute?
  - Unusual computational paradigms.
  - Proteins: do they work like electronic circuits?
  - Genes: what kind of software is that?

- Signaling networks
  - Clearly "information processing"
  - They are "just chemistry": molecule interactions
  - But what are their principles and algorithms?

- Complex, higher-order interactions
  - MAPKKK = MAP Kinase Kinase Kinase:
    that which operates on that which operates on that which operates on protein.

- General models of biological computation
  - What are the appropriate ones?





Ultrasensitivity in the mitogen-activated protein cascade, Chi-Ying F. Huang and James E. Ferrell, Jr., 1996, *Proc. Natl. Acad. Sci. USA*, 93, 10078-10083.

Theory of Computation

- Alan Turing
  - Defined what it means for a problem to be “computable”.
  - Showed that deciding weather an arbitrary mathematical conjecture is true or false is *not computable* (shocking mathematicians). (1936)
  - Also introduced the notion of “universal computation” (now called Turing Completeness): a *single machine* can be built that can compute *any* computable problem. We now call it a computer.
  - These were results in Mathematical Logic, but eventually established Computer Science as a separate discipline.

- John von Neumann
  - Was involved in the design of early electronic computers. The so-called von Newman architecture is at the basis of most computers from the 50’s on.
  - The von Neumann architecture is now seen as a liability: it is strictly sequential and arguably does not make good use of the massive concurrency of electronic hardware. (C.f. massive concurrency of biological systems.)
  - He also developed the foundations of Automata Theory (including cellular automata and robotic self-replication).

Theory of Concurrency

- Early Automata Theory
  - Either about single isolated automata, or about “synchronous” homogeneous collections of automata, like cellular automata.
  - But what about multiple heterogeneous automata talking to each other? This question led to two major developments:

- Petri Nets
  - Dedicated to the study of *causality* relationships between *events*.
  - Providing a basic mathematical model with rich analytical techniques.

- Process Algebra
  - Dedicated to the study of concurrent, nondeterministic, *reactive systems*.
  - Endowing *concurrent languages* with a mathematical semantics.
  - Can provide foundations and inspiration for molecular programming, because molecular interactions are massively concurrent and heterogeneous.

Reactive Systems

- A complex system does not compute a function
  - What function does E-coli compute?
  - Organisms, operating systems, computer networks, do not compute functions: they *indefinitely* react to stimuli and hold *internal state*.
  - Hence we need a mathematical treatment of *interactions*, not of *functions*. This has long been recognized and addressed in Computer Science.

- Reactive Systems
  - A system of components that each *react* to other components.
  - Each component is independently described in terms of its reactions to stimuli (which may come from many different other components).
  - The behavior of the system emerges from the free interactions of the components.
  - No a-priori description of all possible states (e.g. all possible molecular complexes) is needed.

Process Algebra

[Hoare, Milner, Pnueli, etc.]

- Reactive systems (living organisms, computer networks, operating systems, …)
  - Math is based on *entities that react/interact with their environment* (“*processes*”), not on functions from domains to codomains.
- Concurrent
  - Events (reactions/interactions) happen concurrently and asynchronously, not sequentially like in function composition.
- Stochastic
  - Or probabilistic, or nondeterministic, but is never about deterministic system evolution.
- Stateful
  - Each concurrent activity (“process”) maintains its own local state, as opposed to stateless functions from inputs to outputs.
- Discrete
  - Evolution through discrete transitions between discrete states, not incremental changes of continuous quantities.
- Kinetics of interaction
  - An “interaction” is anything that moves a system from one state to another.

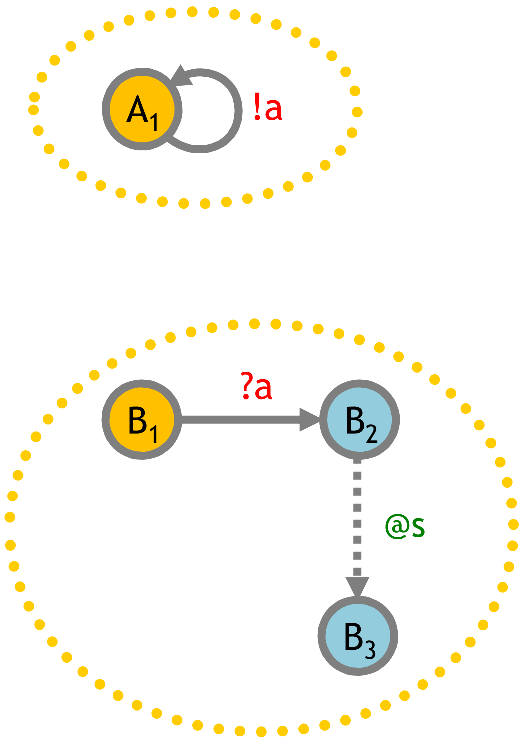
Part I: From Molecules to Automata

(Macro-) Molecules as (Interacting) Automata

Interacting Automata

Legend

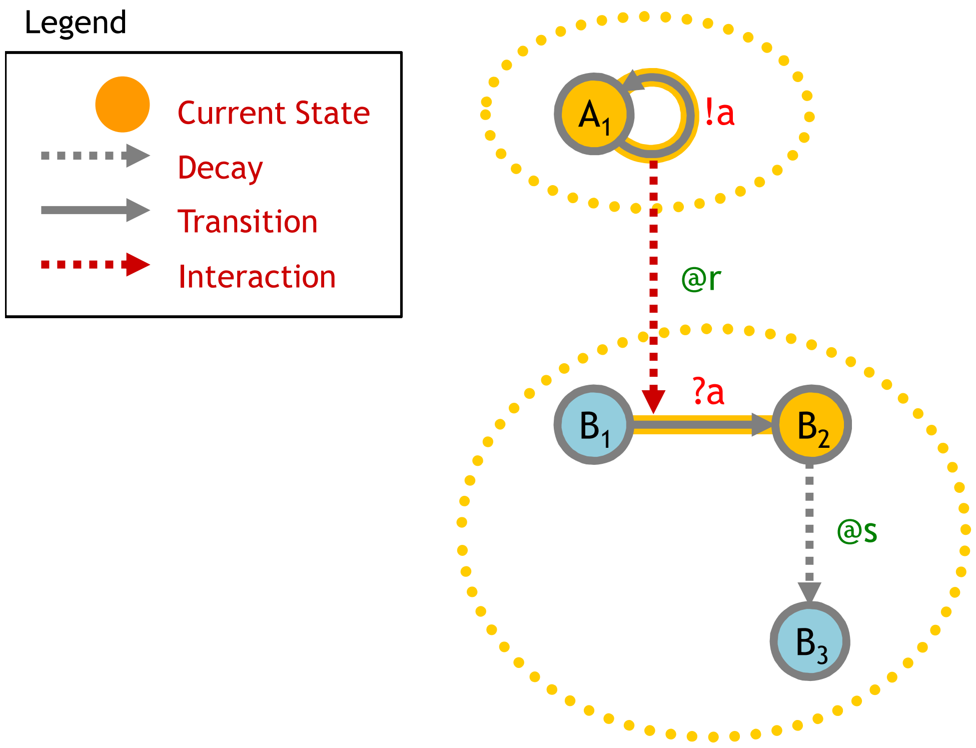
- ● Current State
- ⇢ Decay
- → Transition
- ⇢ Interaction



A_1	is a *state*
a	is a *channel* i.e. a named *interaction interface* (e.g. a surface patch)
?,!	indicate any *complementarity* of interaction (e.g. charge)
?a, !a	indicate *complementary actions*,
@r, @s	are rates

Kinetic laws:

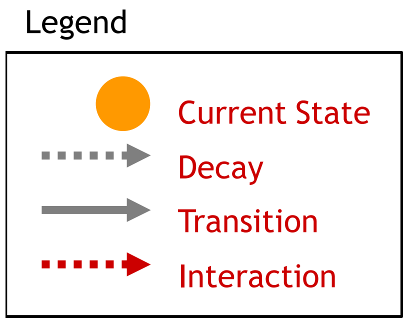
Interacting Automata

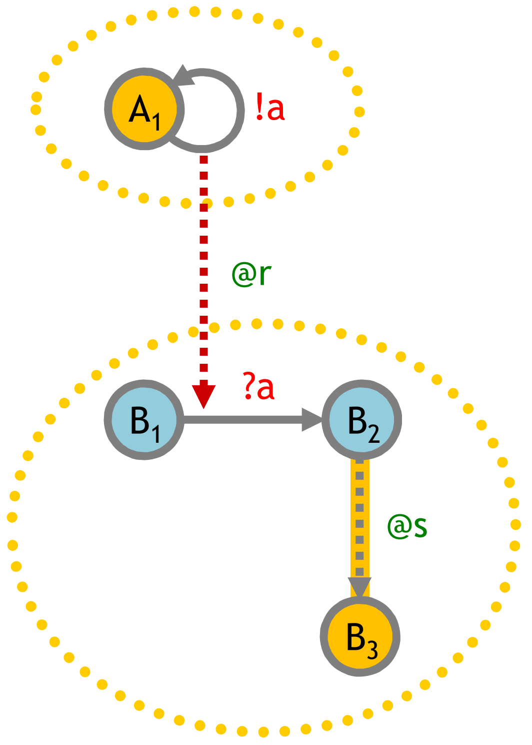


A_1	is a *state*
a	is a *channel* i.e. a named *interaction interface* (e.g. a surface patch)
?,!	indicate any *complementarity* of interaction (e.g. charge, shape)
?a, !a	indicate *complementary actions*, joined by an interaction arrow ⇢
@r, @s	are rates

Kinetic laws: ***Two complementary actions may result in an interaction.***

Interacting Automata

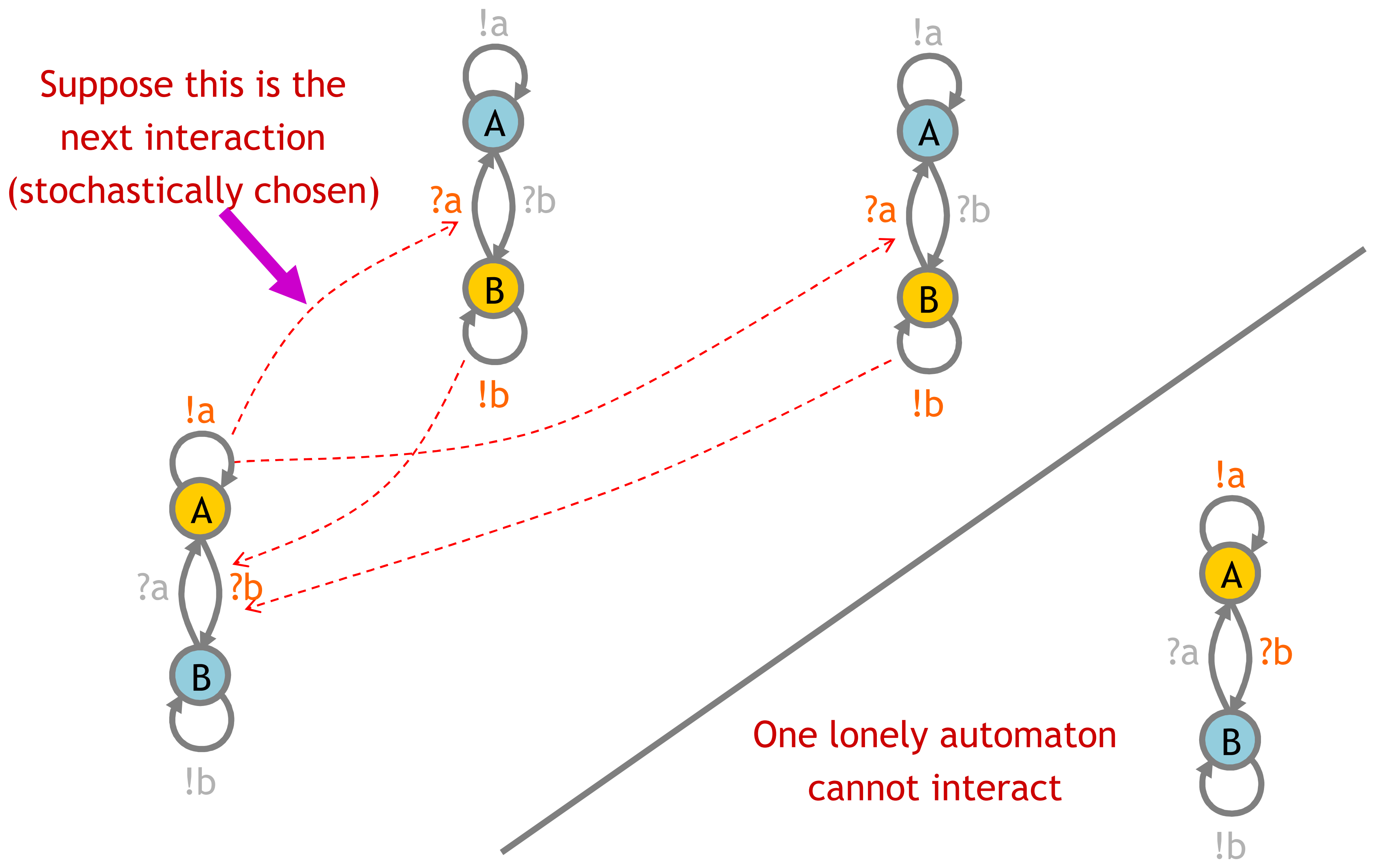




A_1	is a *state*
a	is a *channel* i.e. a named *interaction interface* (e.g. a surface patch)
?,!	indicate any *complementarity* of interaction (e.g. charge)
?a, !a	indicate *complementary actions*, joined by an interaction arrow ···▶
@r, @s	are rates

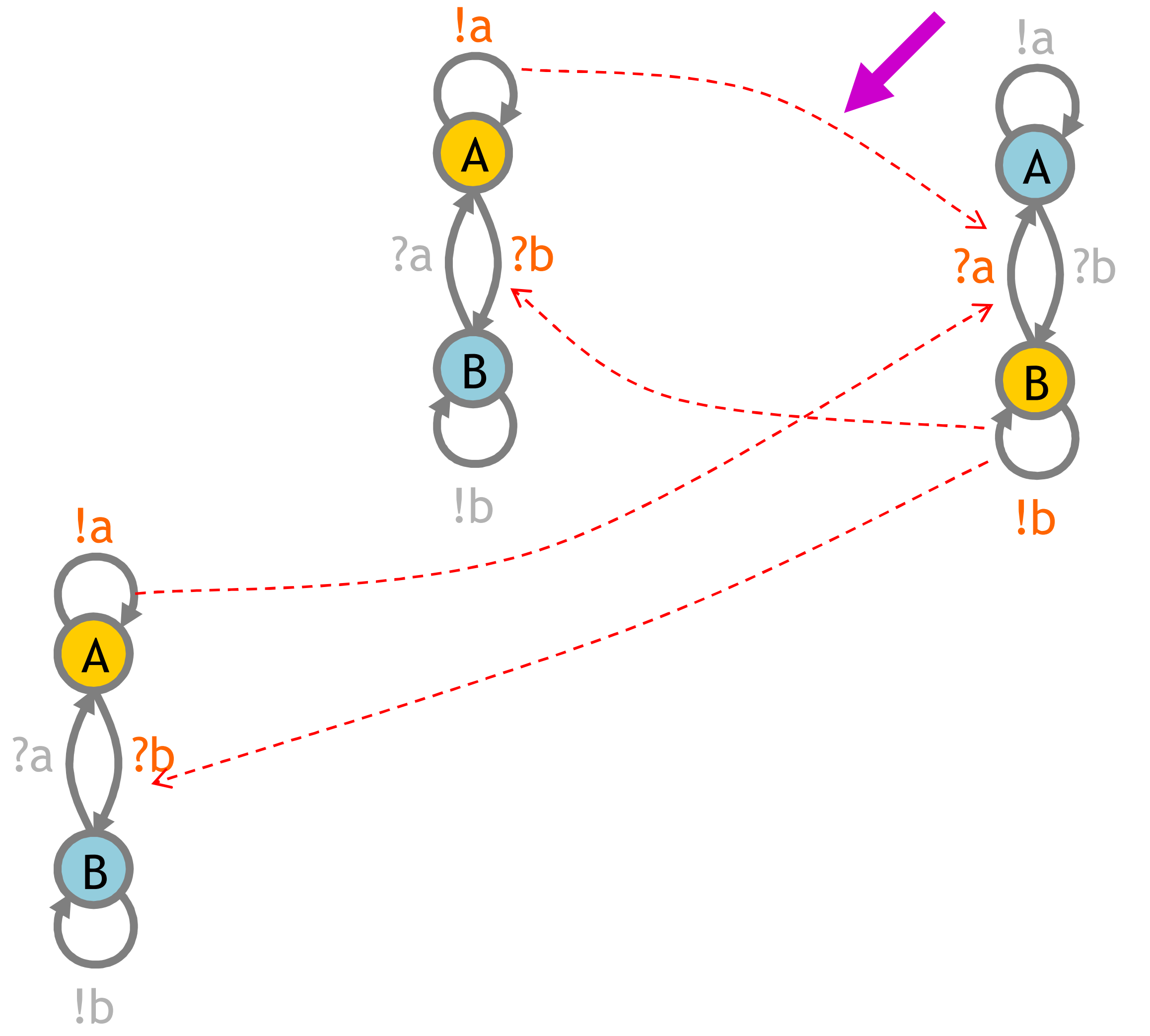
Kinetic laws: ***Two complementary actions may result in an interaction.*** ***A decay may happen spontaneously.***

Interactions in a Population

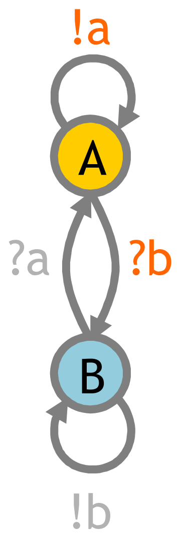


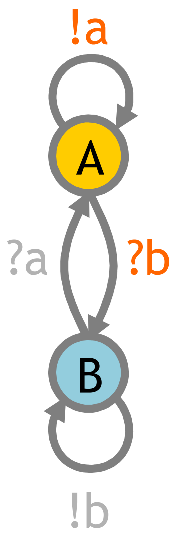
Suppose this is the next interaction (stochastically chosen)
!a
A
?a
?b
B
!b
!a
A
?a
?b
B
!b
!a
A
?a
?b
B
!b
One lonely automaton cannot interact
!a
A
?a
?b
B
!b

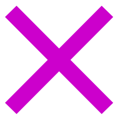
Interactions in a Population

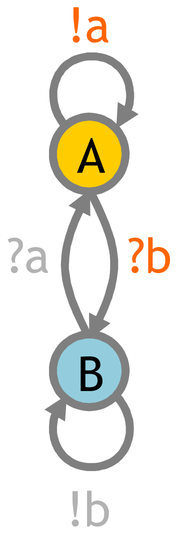


Interactions in a Population



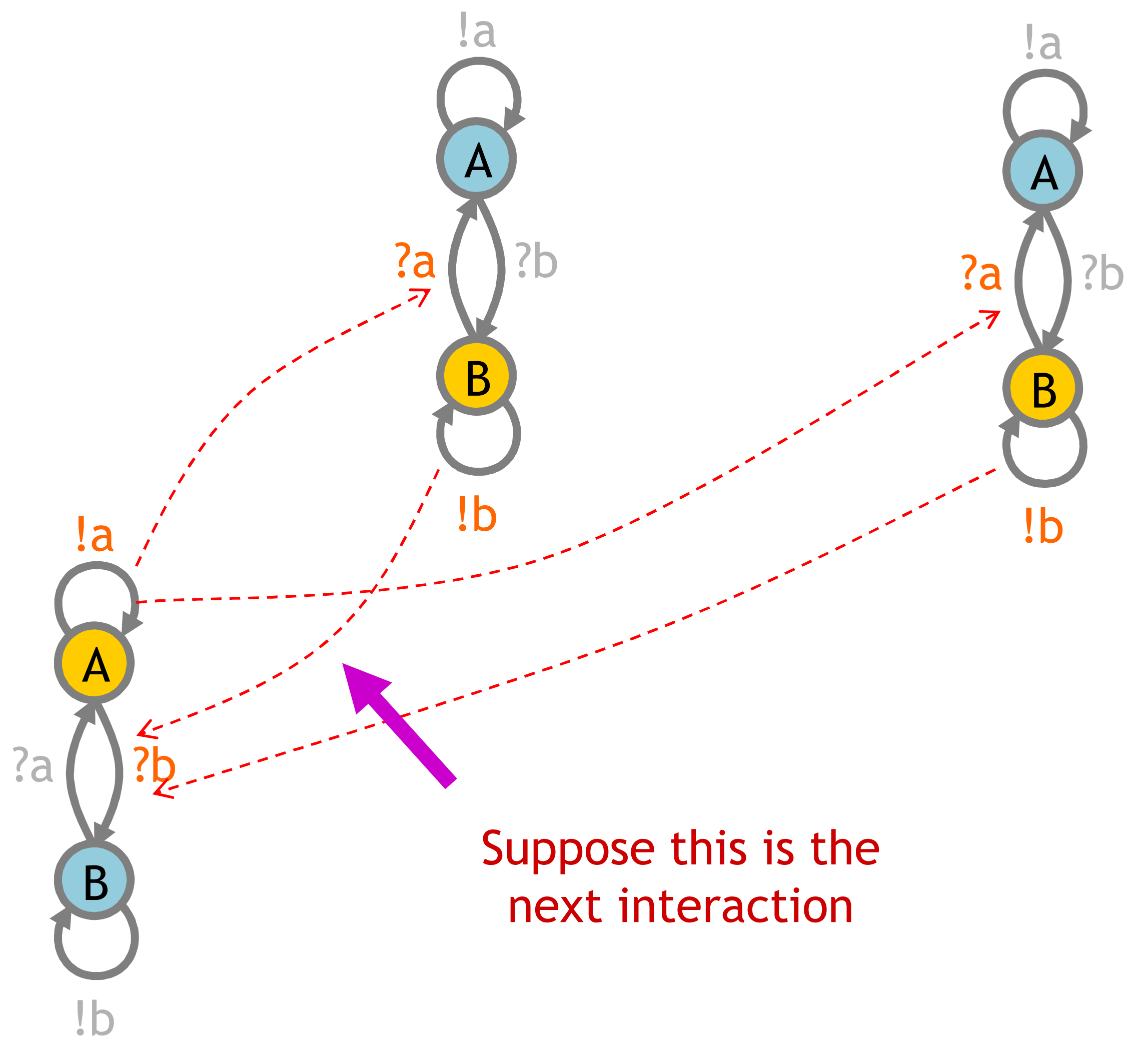






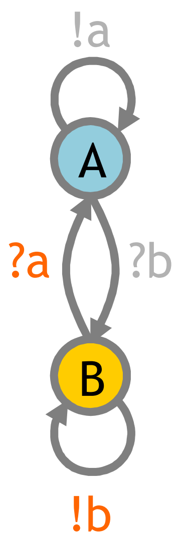
All-A stable
population

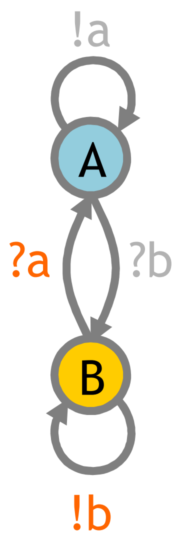
Interactions in a Population (2)

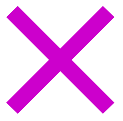


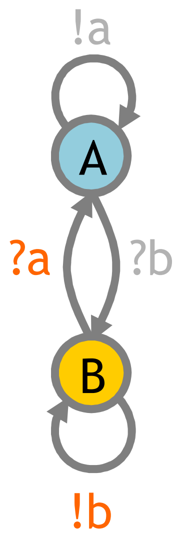
Suppose this is the next interaction

Interactions in a Population (2)





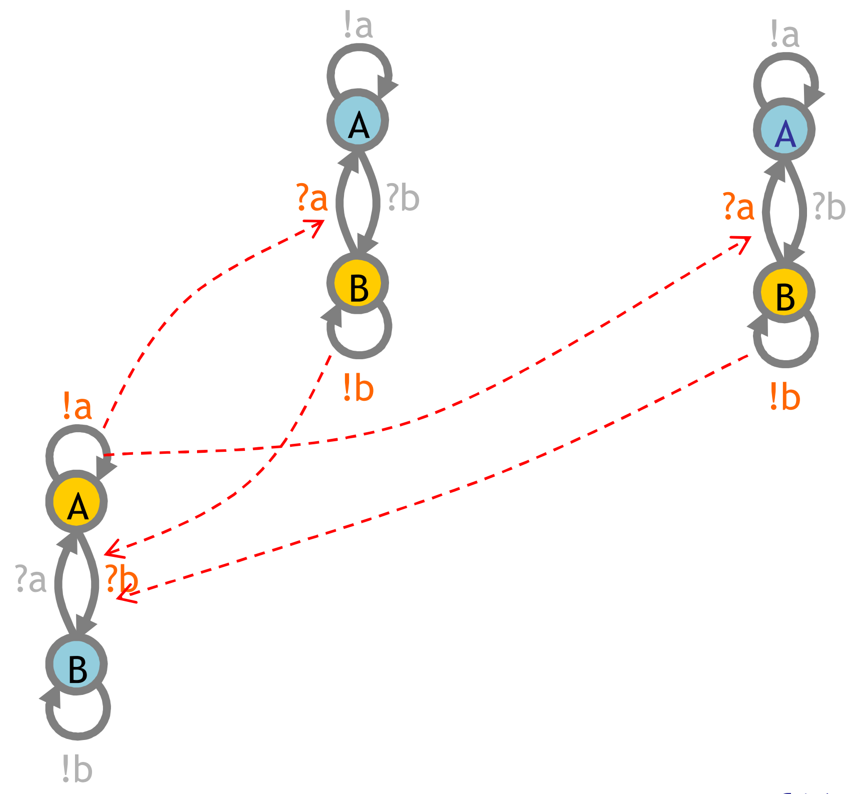


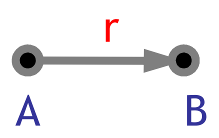


All-B stable population

Nondeterministic population behavior ("multistability")

CTMC Semantics



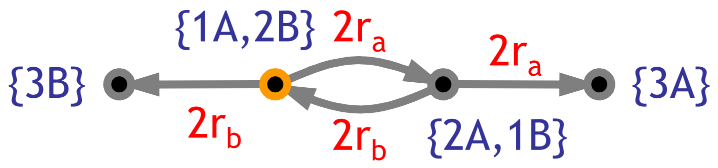


CTMC
(homogeneous) Continuous Time Markov Chain
- directed graph with no self loops
- nodes are system states
- arcs have transition rates

Probability of holding in state A:

$Pr(H_A>t) = e^{-rt}$

in general, $Pr(H_A>t) = e^{-Rt}$ where R is the sum of all the exit rates from A



CTMC

Reactions vs. Components

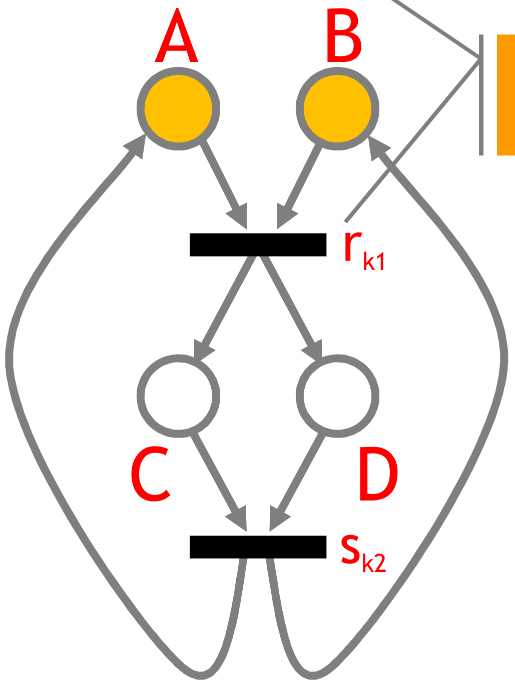
Says what "A" *does*.

Says what "A" *is*.

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

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

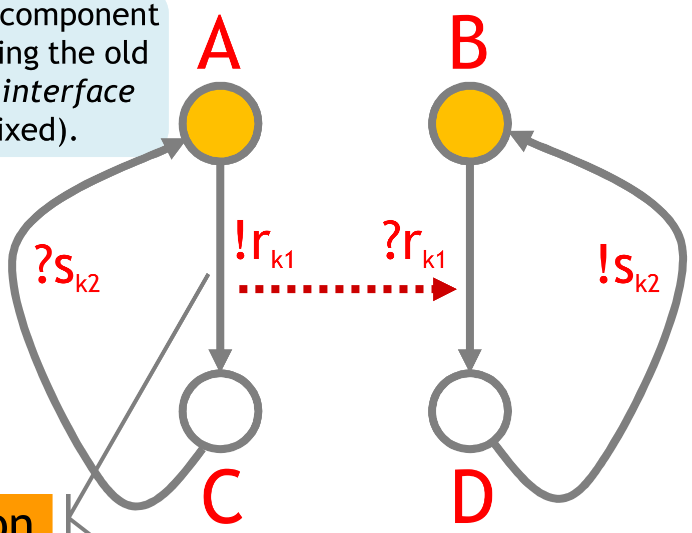
Does A become C or D?



Reaction oriented

1 line per reaction

Can add a new component without changing the old ones (if their *interface* remains fixed).



Interaction oriented

1 line per component

$A = !r_{k1}; C$

$C = ?s_{k2}; A$

$B = ?r_{k1}; D$

$D = !s_{k2}; B$

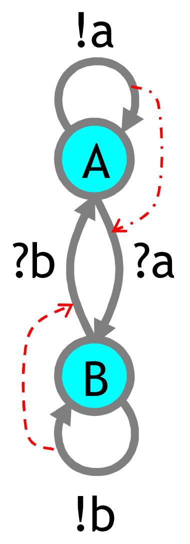
A becomes C not D!

The same "state space"

$\boxed{\text{CTMC}}$

Groupies and Celebrities

Groupies and Celebrities



Celebrity

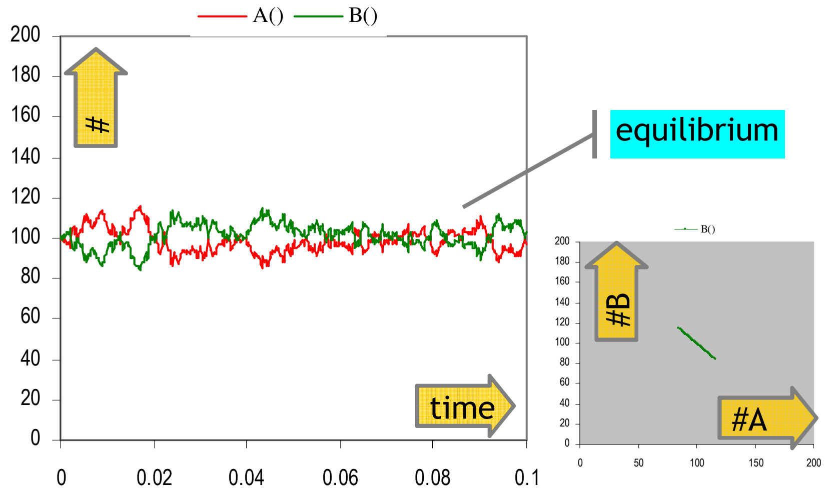
(does not want to be like somebody else)

```
directive sample 1.0 1000
directive plot A(); B()

new a@1.0:chan()
new b@1.0:chan()

let A() = do !a; A() or ?a; B()
and B() = do !b; B() or ?b; A()

run 100 of (A() | B())
```

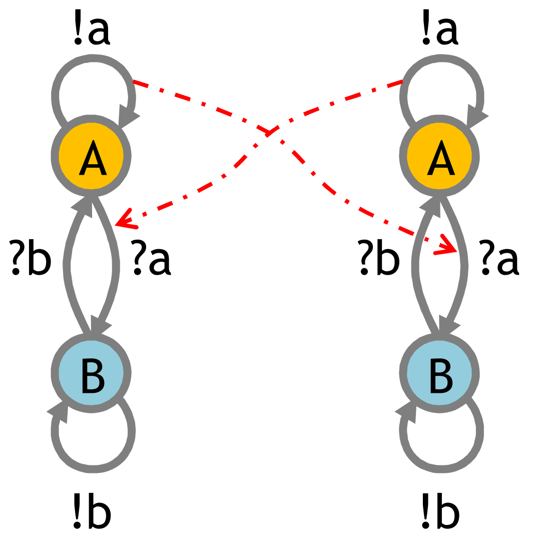
a@1.0

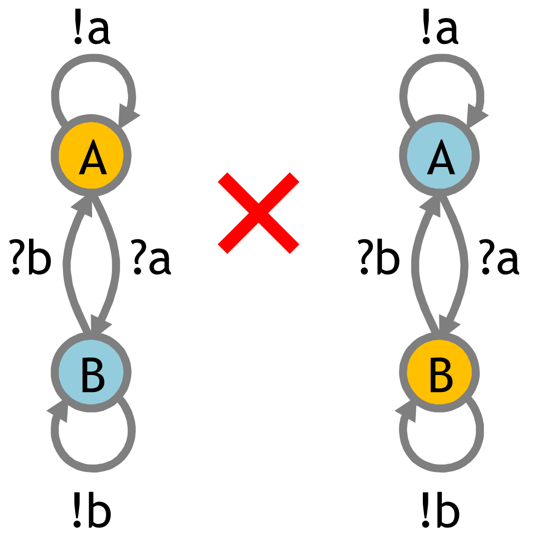
b@1.0

A stochastic collective of celebrities:

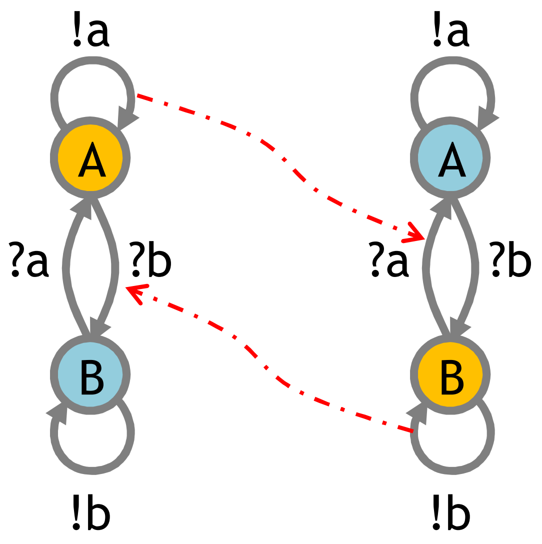


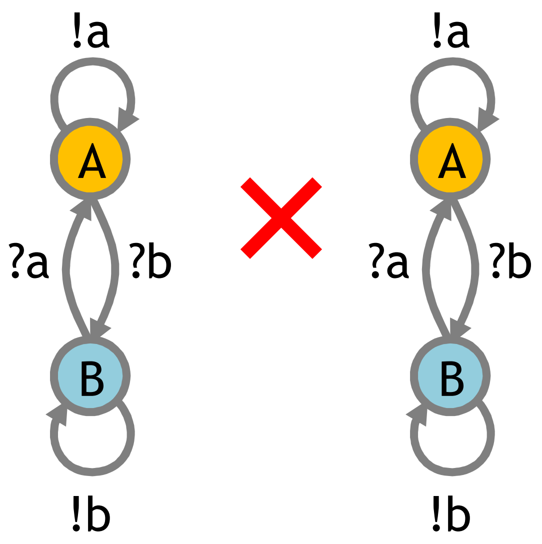
Stable because as soon as a A finds itself in the majority, it is more likely to find somebody in the same state, and hence change, so the majority is weakened.

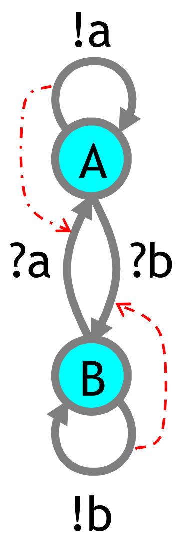




Groupies and Celebrities







Groupie

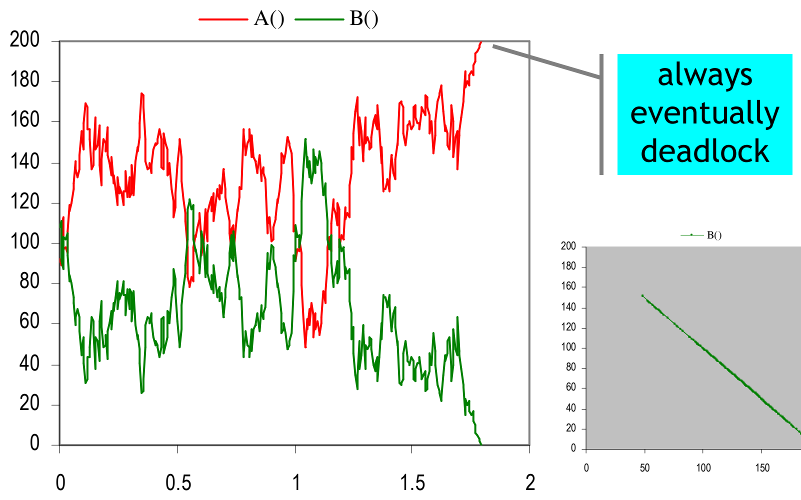
(wants to be like somebody different)

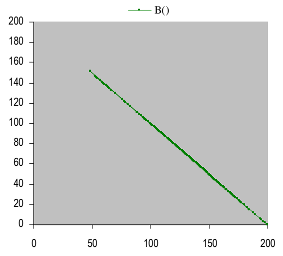
```
directive sample 1.0 1000
directive plot A(); B()

new a@1.0:chan()
new b@1.0:chan()

let A() = do !a; A() or ?b; B()
and B() = do !b; B() or ?a; A()

run 100 of (A() | B())
```

a@1.0

b@1.0

A stochastic collective of groupies:



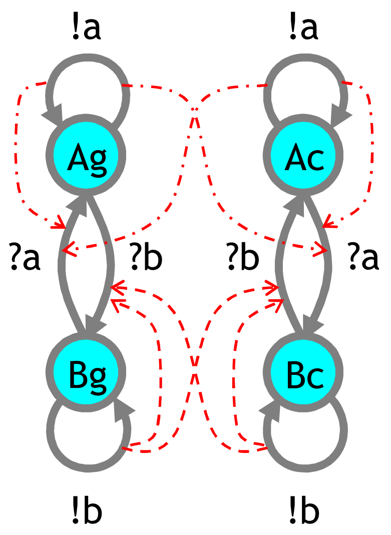


Unstable because within an A majority, an A has difficulty finding a B to emulate, but the few B's have plenty of A's to emulate, so the majority may switch to B. Leads to deadlock when everybody is in the same state and there is nobody different to emulate.

Both Together

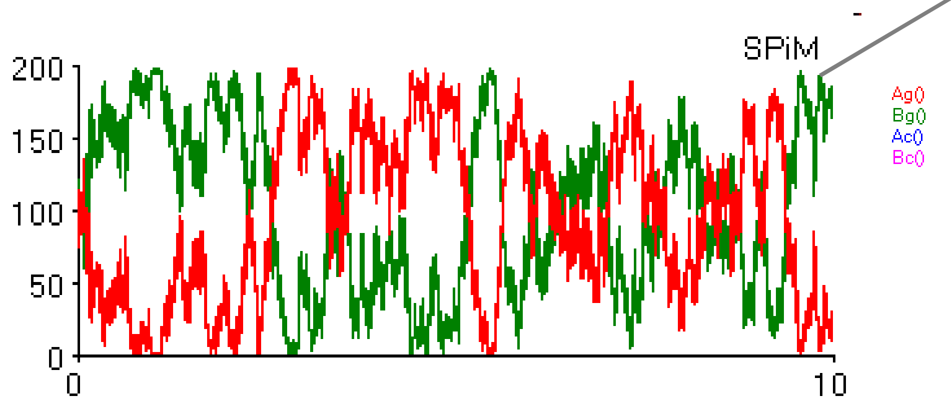
A way to break the deadlocks: Groupies with just a few Celebrities

Many Groupies



A few Celebrities

```
directive sample 10.0
directive plot Ag(); Bg(); Ac(); Bc()

new a@1.0:chan()
new b@1.0:chan()

let Ac() = do !a; Ac() or ?a; Bc()
and Bc() = do !b; Bc() or ?b; Ac()

let Ag() = do !a; Ag() or ?b; Bg()
and Bg() = do !b; Bg() or ?a; Ag()

run 1 of Ac()
run 100 of (Ag() | Bg())
```

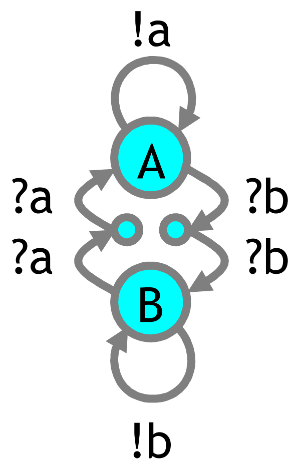


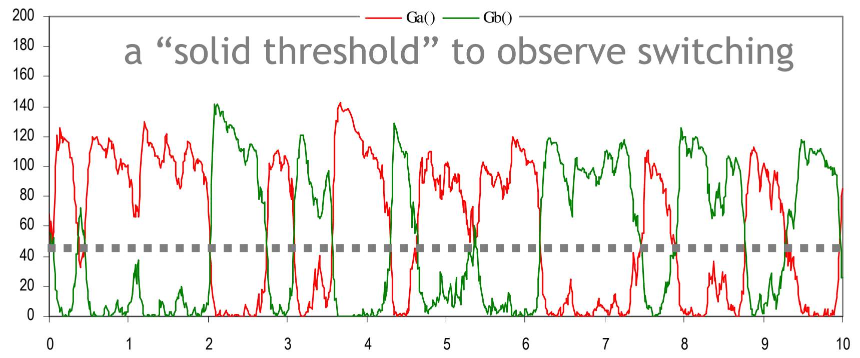
never deadlock

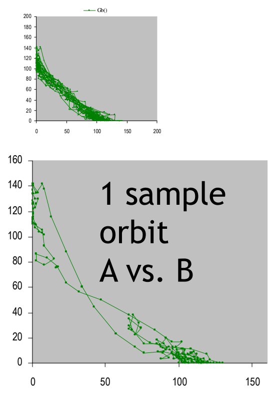
A tiny bit of "noise" can make a huge difference

Hysteric Groupies

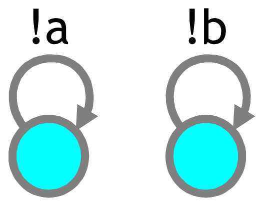
We can get more regular behavior from groupies if they “need more convincing”, or “hysteresis” (history-dependence), to switch states.







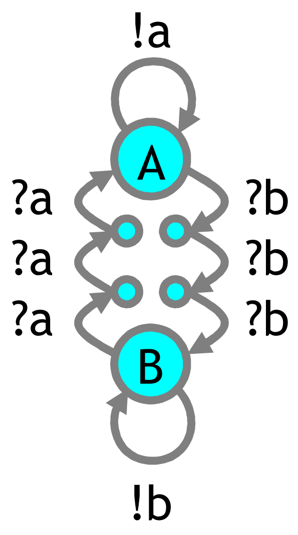
```
directive sample 10.0 1000
directive plot Ga(); Gb()

new a@1.0:chan()
new b@1.0:chan()

let Ga() = do !a; Ga() or ?b; ?b; Gb()
and Gb() = do !b; Gb() or ?a; ?a; Ga()

let Da() = !a; Da()
and Db() = !b; Db()

run 100 of (Ga() | Gb())
run   1 of (Da() | Db())
```

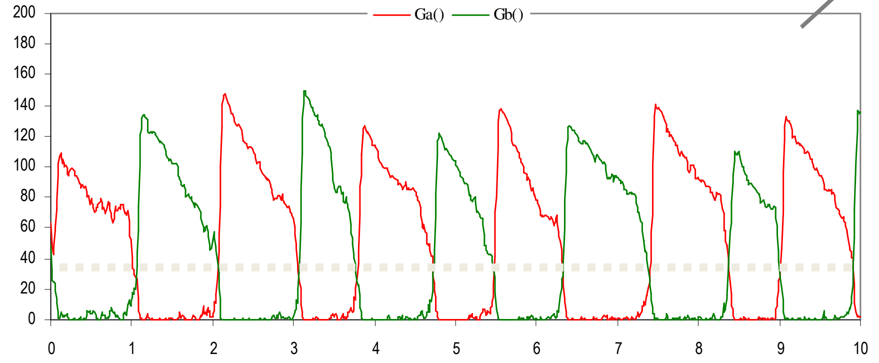


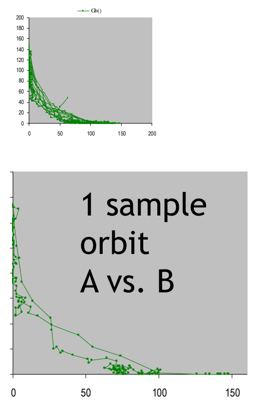
(With doping to break deadlocks)

N.B.: It will not oscillate without doping (noise)

“regular” oscillation



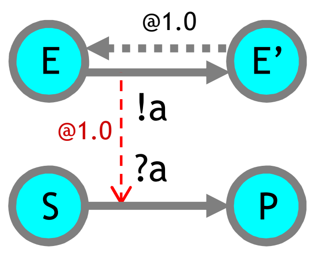


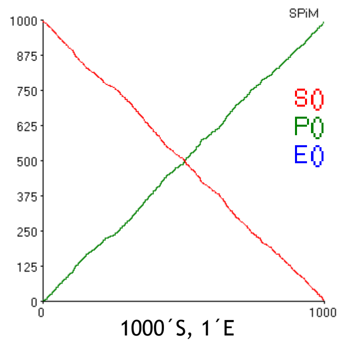


```
directive sample 10.0 1000
directive plot Ga(); Gb()

new a@1.0:chan()
new b@1.0:chan()

let Ga() = do !a; Ga() or ?b; ?b; ?b; Gb()
and Gb() = do !b; Gb() or ?a; ?a; ?a; Ga()

let Da() = !a; Da()
and Db() = !b; Db()

run 100 of (Ga() | Gb())
run   1 of (Da() | Db())
```

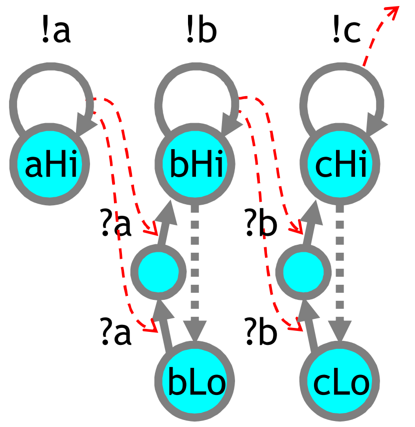
Some Devices

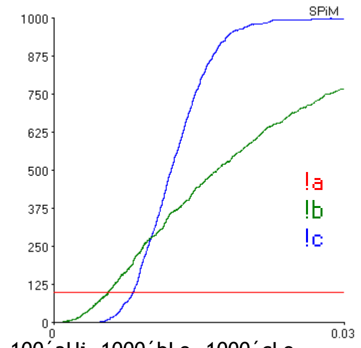
Linear Pump





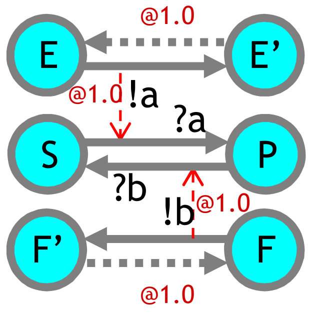
Cascade Amplifier

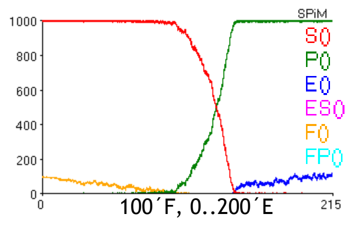




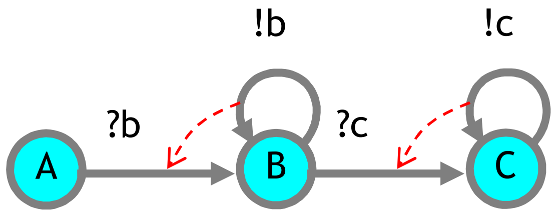
100´aHi, 1000´bLo, 1000´cLo, rates=1.0

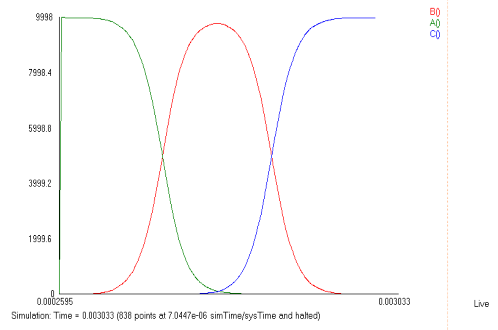
Ultrasensitive Switch





Symmetric Wave Generator

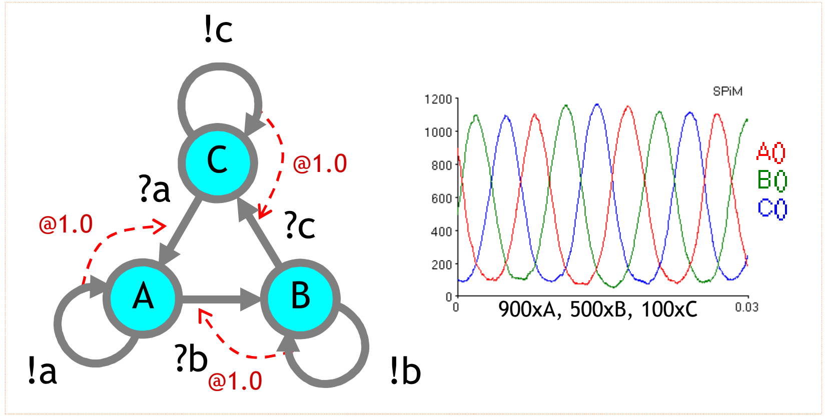




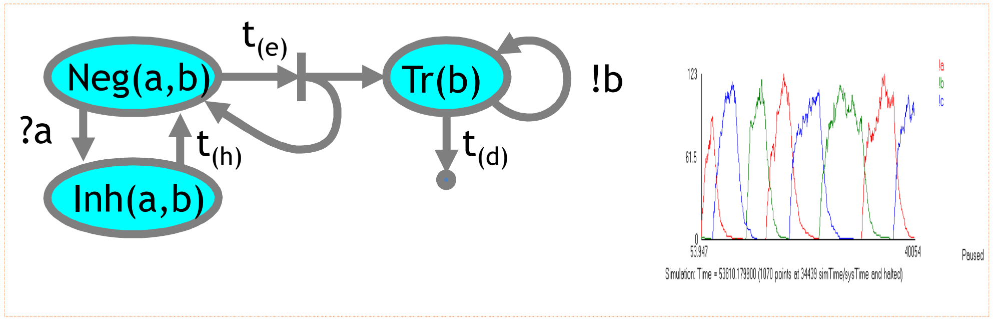
Simulation: Time = 0.003033 (838 points at 7.0447e-06 simTime/sysTime and halted)

More Devices

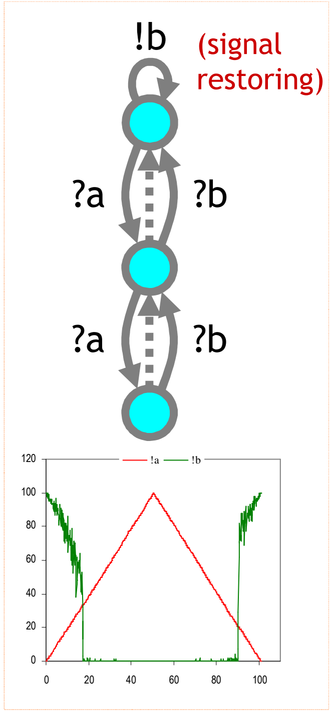
Oscillator



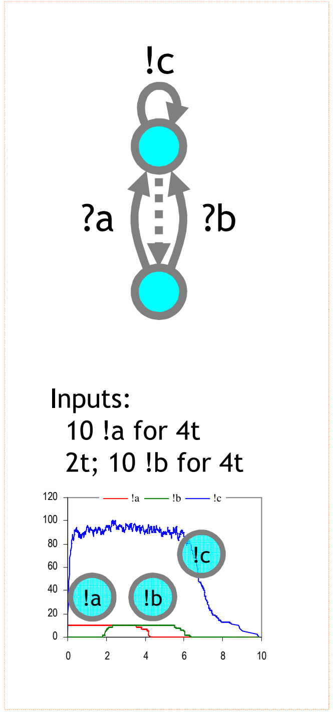
Repressilator (1 of 3 similar gates)



b = not a

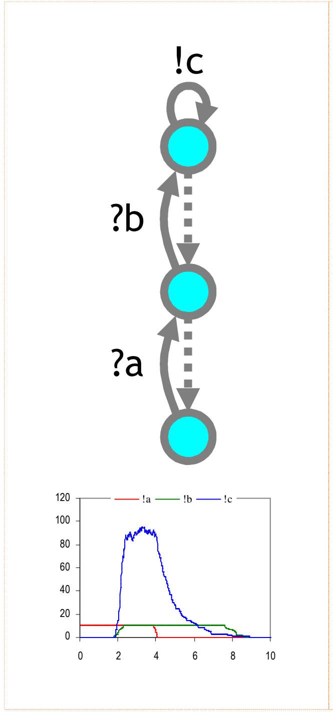


c = a or b

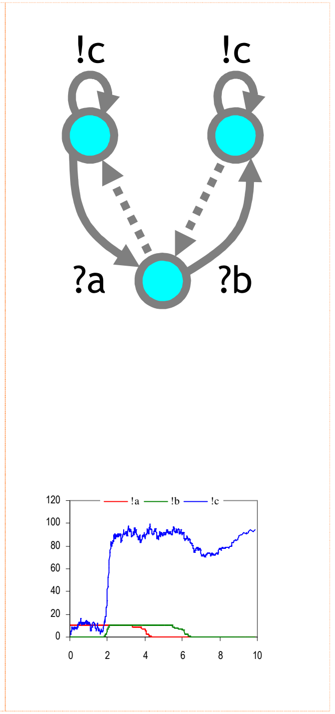


Inputs:
10 !a for 4t
2t; 10 !b for 4t

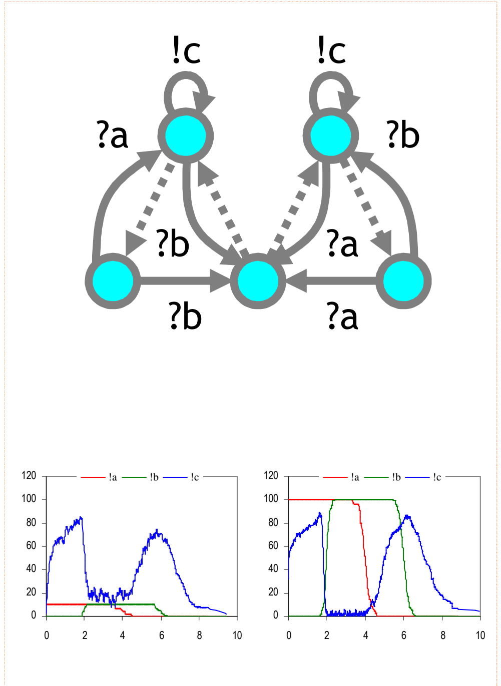
c = a and b



c = a imply b

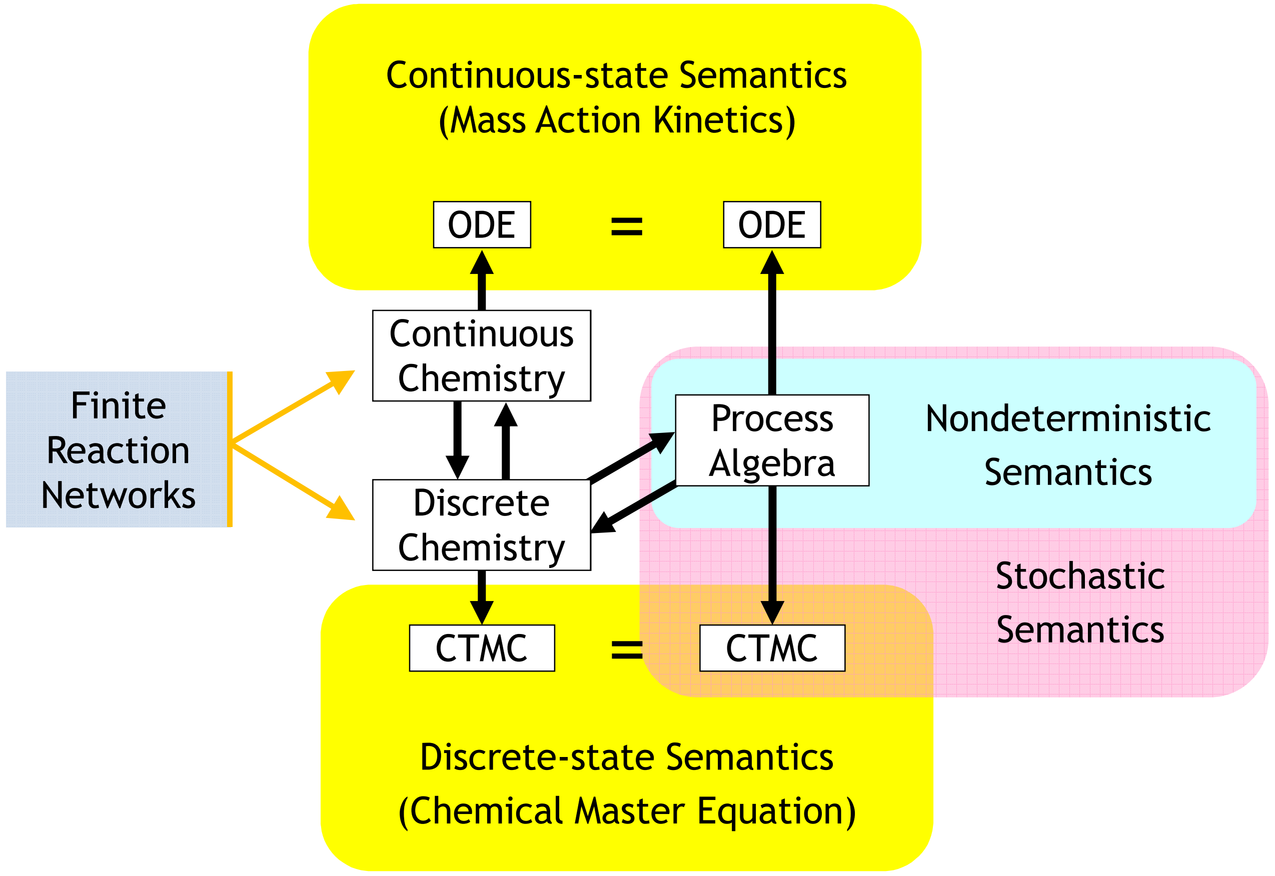


c = a xor b



Semantics of Collective Behavior

The Two Semantic Sides of Chemistry

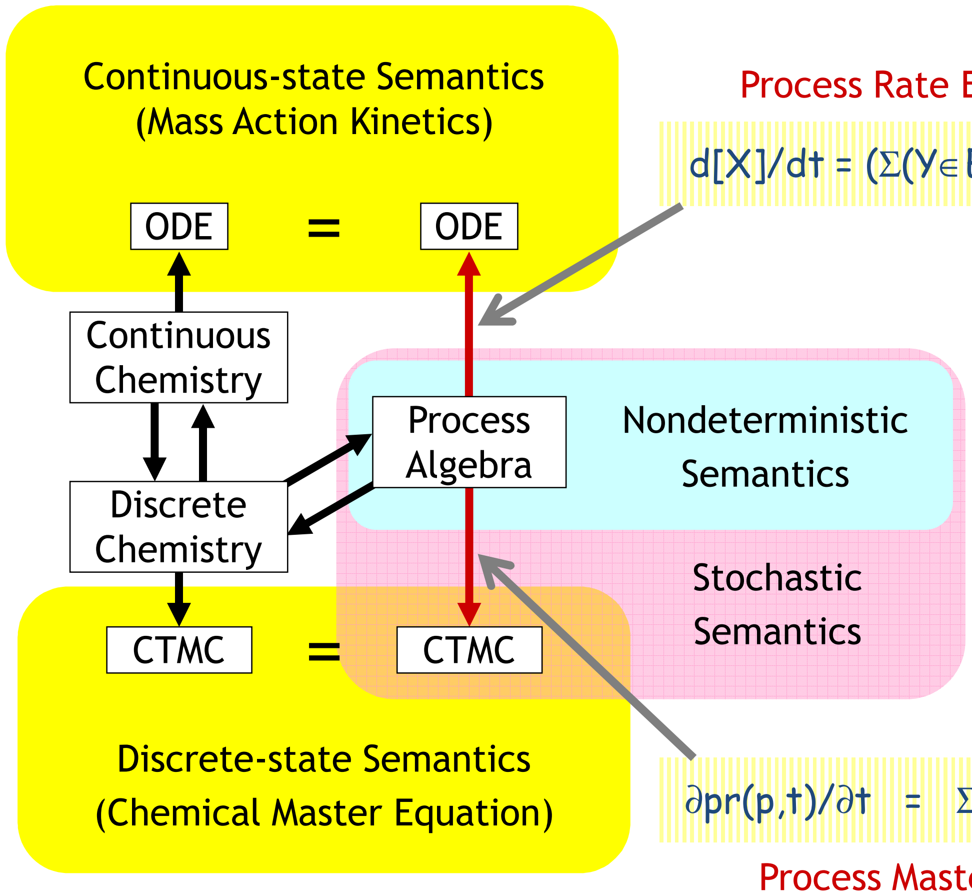


These diagrams commute via appropriate maps.

L. Cardelli: “On Process Rate Semantics” (TCS)

L. Cardelli: “A Process Algebra Master Equation” (QEST’07)

Quantitative Process Semantics



Process Rate Equation

$d[X]/dt = (\Sigma(Y\in E)\ Accr_E(Y,X)\cdot[Y]) - Depl_E(X)\cdot[X]$ for all $X\in E$

Accretion Depletion

Defined over the syntax of processes

Interactions Propensity

$\partial pr(p,t)/\partial t = \Sigma_{i\in\mathfrak{I}}\ a_i(p-v_i)\cdot pr(p-v_i,t) - a_i(p)\cdot pr(p,t)$ for all $p\in States(E)$

Process Master Equation

Process Algebra
Beyond Finite Reaction Networks
(with Gianluigi Zavattaro)

Turing Completeness

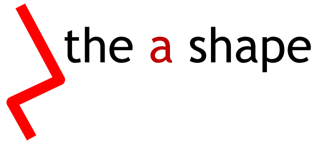
- Turing Completeness
  - A Turing Machine is “universal”: it can emulate any other computing device.
  - Your laptop is similarly a universal computing device.
  - Is chemistry universal: can chemistry emulate any computing device?
- Finite Reaction Networks are equivalent to Petri Nets.
  - It is possible to translate any finite system of chemical reactions into a Place/Transition Petri Net (ignoring rates). Reachability of a dead (“halting”) state in P/T nets is decidable (an algorithm can answer yes/no).
  - By Turing’s theorem, if termination is decidable, i.e. if it is a simple problem, then the computational system is not universal. In particular, it cannot emulate a Turing machine (or your laptop).
- Hence finite reaction networks are not Turing-complete (Soloveichik et. al., Natural Computing 2008)
  - Finite chemistry can’t compute!
  - Even though finite stochastic chemistry includes, e.g., chaotic systems.
  - However, finite *stochastic* reaction networks can *approximate* Turing machines to any precision (slowly).

“Turifying” Chemistry

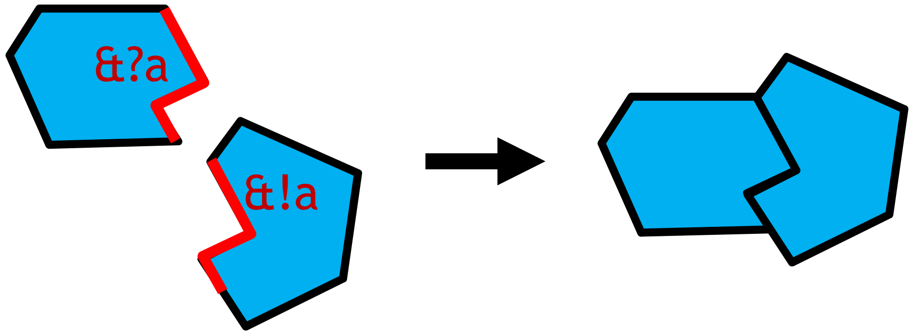
- Interacting Automata are not Turing complete
  - They are equivalent to finite reaction networks, and to Petri Nets.
- What can we add to achieve Turing completeness?
  - It is not easy to add power to finite reaction networks, other than making them *infinite* (hence ‘non-programs’).
  - But it is easy to add power to simple automata, while keeping them *finite*.
  - E.g. we can go to standard process algebras, which are finite programming languages and are Turing complete.
- But is there…
  - A *basic* extension mechanism
  - which is also biologically *realistic*?

Association and Dissociation

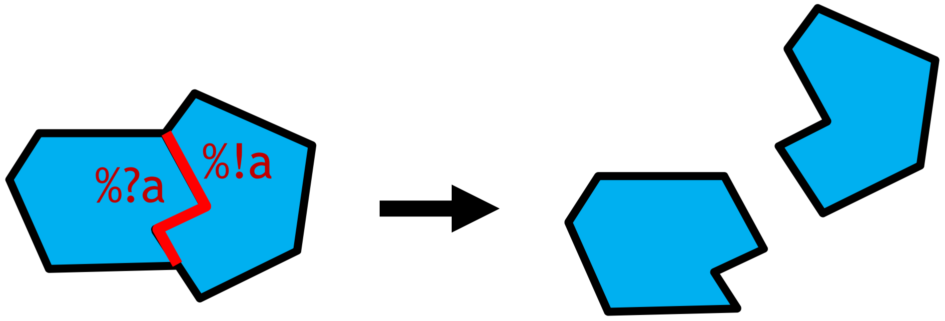
- Association patches are named



- & – association
  - &?a associate
  - &!a co-associate



- % – dissociation
  - %?a dissociate
  - %!a co-dissociate



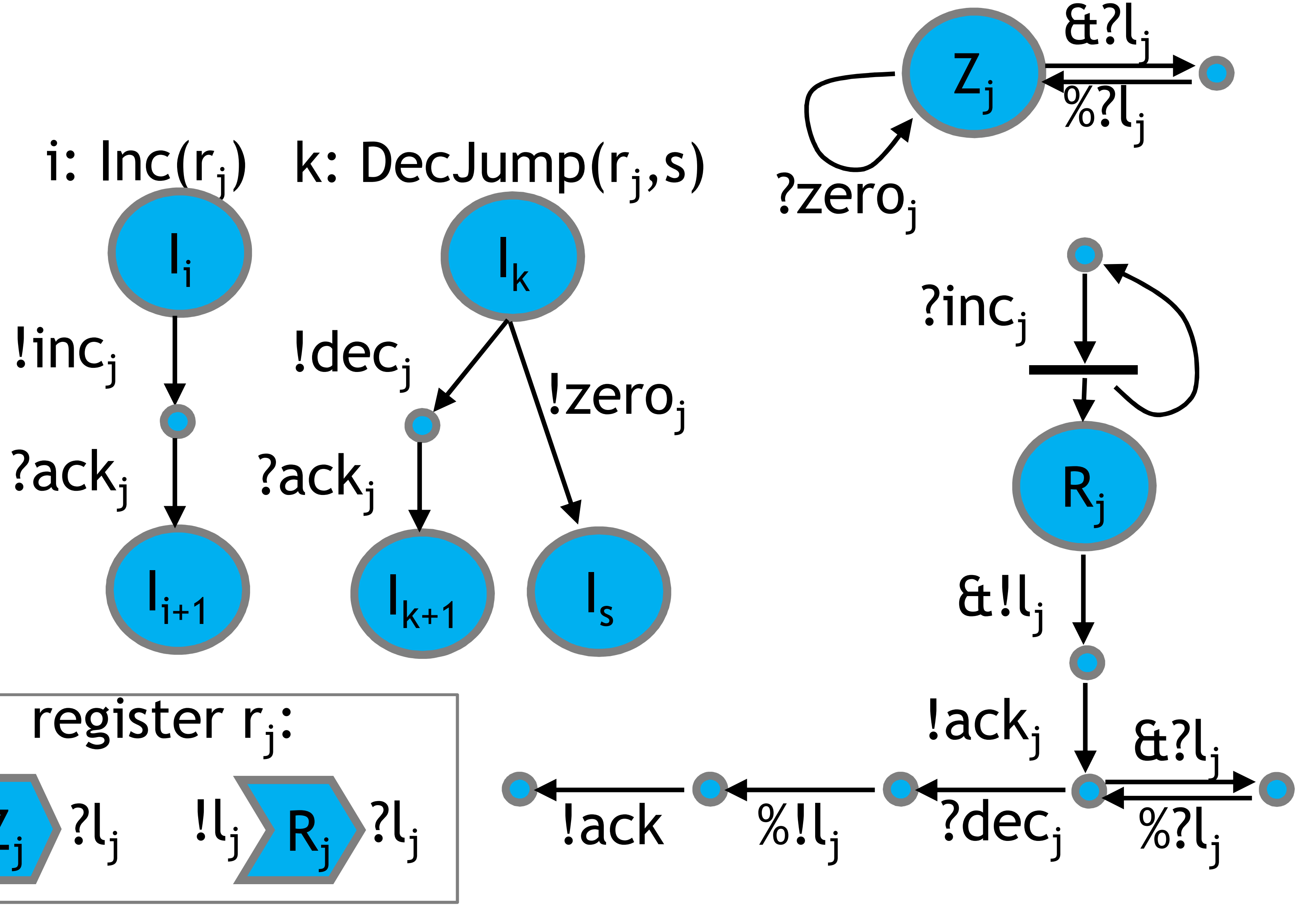
- A given patch can *hold* only one association at a time
- Two molecules can dissociate only if *they* are associated

Turing completeness of "Biochemistry"

- Random Access Machines: [Min67]
  - **Registers**: r_1 … r_n hold natural numbers (unbounded)
  - **Program**: finite sequence of numbered instructions
    - **i: Inc(r_j)**: add 1 to the content of r_j and go to the next instruction
    - **i: DecJump(r_j,s)**: if the content of r_j is not 0 then decrease by 1 and go to the next instruction; otherwise jump to instruction s

- There is a RAM encoding in BGF (= automata with complexation)
  - Hence BGF is Turing complete.
  - Removing the old "collision" interactions keep it Turing complete (they can be expressed by association/dissociation).
  - But removing association *or* dissociation makes it non Turing complete.

RAM encoding in BGF



Why is This Easier in Process Algebra?

Example: Linear Polymerization

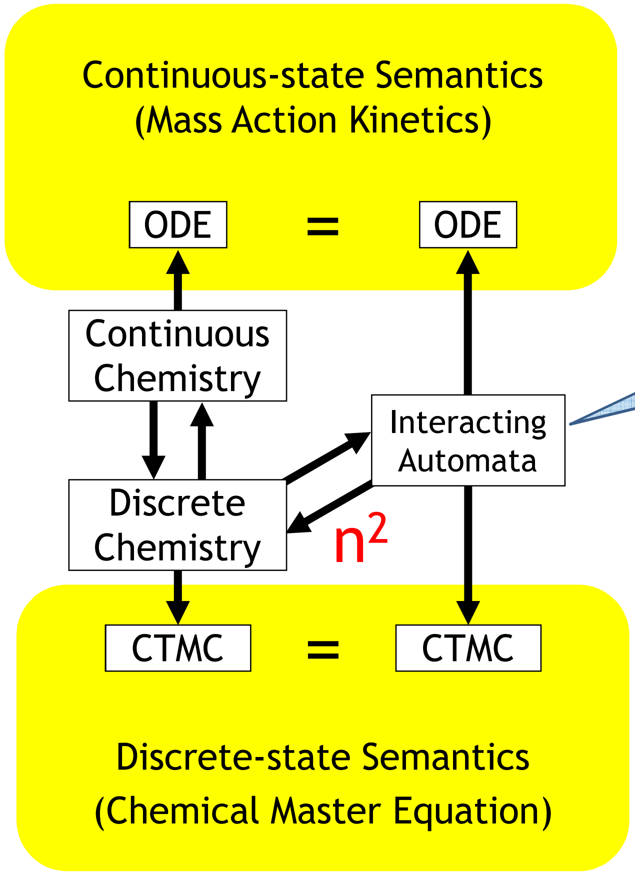
- In chemistry we have to write an *infinite list of reactions*
  - $P_0 + M \rightarrow P_1$
  - $P_1 + M \rightarrow P_2$
  - etc.
  - An infinite list of things is *not* a computation device!
    And the specificity of an infinite set of reactions unrealistic.

- In process algebra we can write a *finite set of interactions*
  - A polymer (of any length) with a free surface, plus a monomer with a complementary surface, gives you another polymer with a free surface.
  - That's it.

- Process algebra descriptions are intrinsically more compact
  - This is an extreme case (finite vs. infinite)
  - But is also true for finite cases (linear vs. quadratic or exponential).

Part I Summary

Process Algebra is 'Bigger' and 'More Compact' than finite chemistry

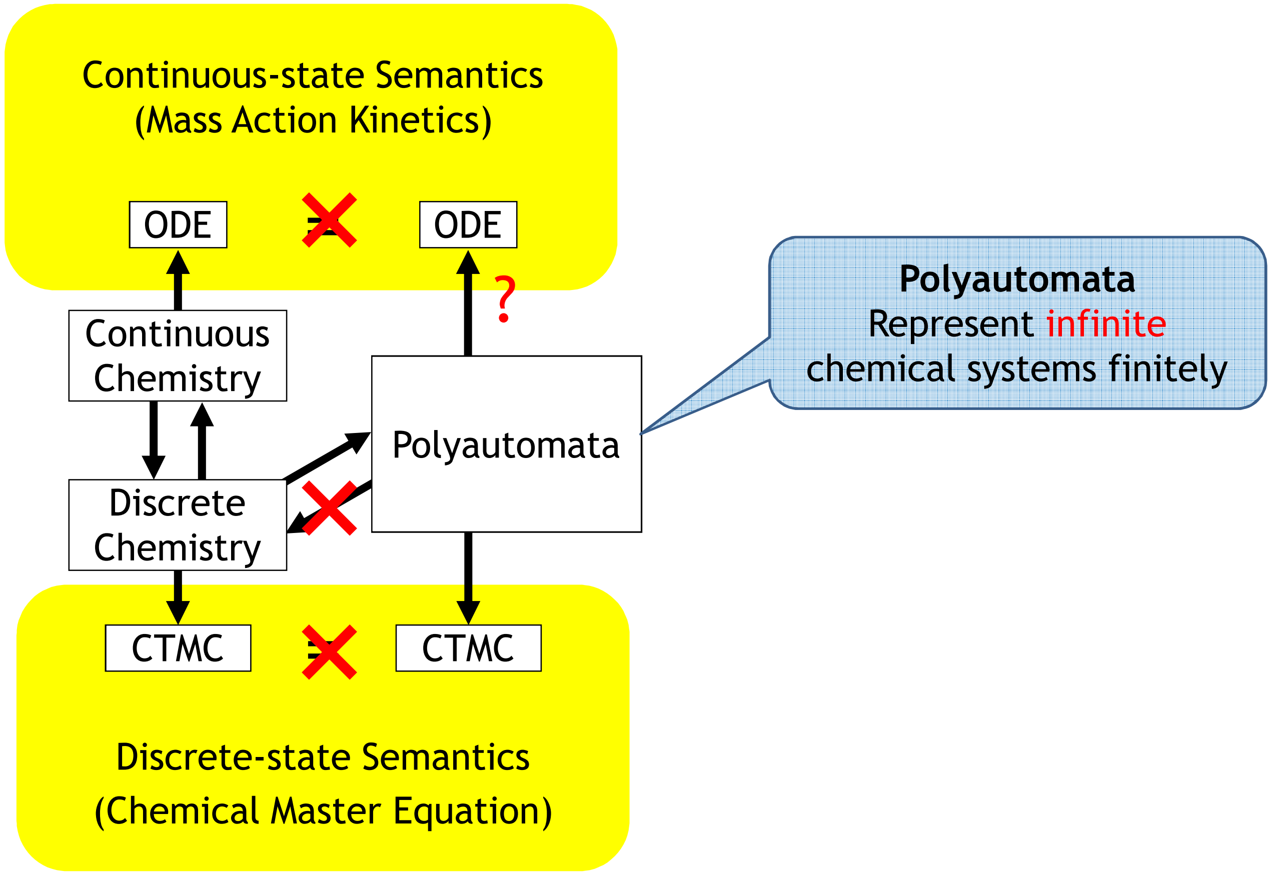


Continuous-state Semantics (Mass Action Kinetics)
ODE
=
ODE
Continuous Chemistry
Interacting Automata
Discrete Chemistry
n^2
CTMC
=
CTMC
Discrete-state Semantics (Chemical Master Equation)

Interacting Automata
Represent finite chemical systems compactly

Part I Summary

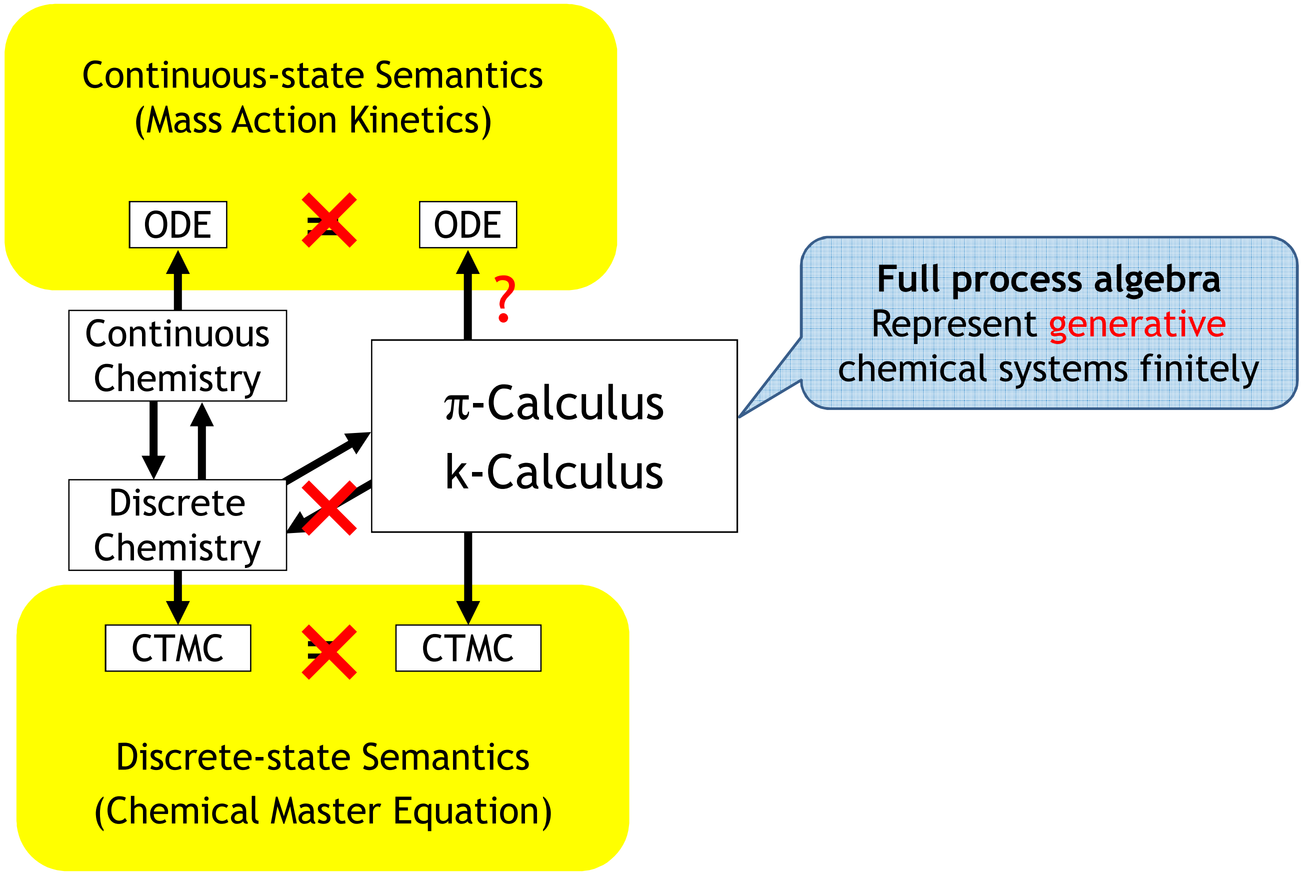
Process Algebra is 'Bigger' and 'More Compact' than finite chemistry



Continuous-state Semantics (Mass Action Kinetics)
ODE
=
ODE
?
Continuous Chemistry
Polyautomata
Discrete Chemistry
CTMC
=
CTMC
Discrete-state Semantics (Chemical Master Equation)
Polyautomata
Represent infinite chemical systems finitely

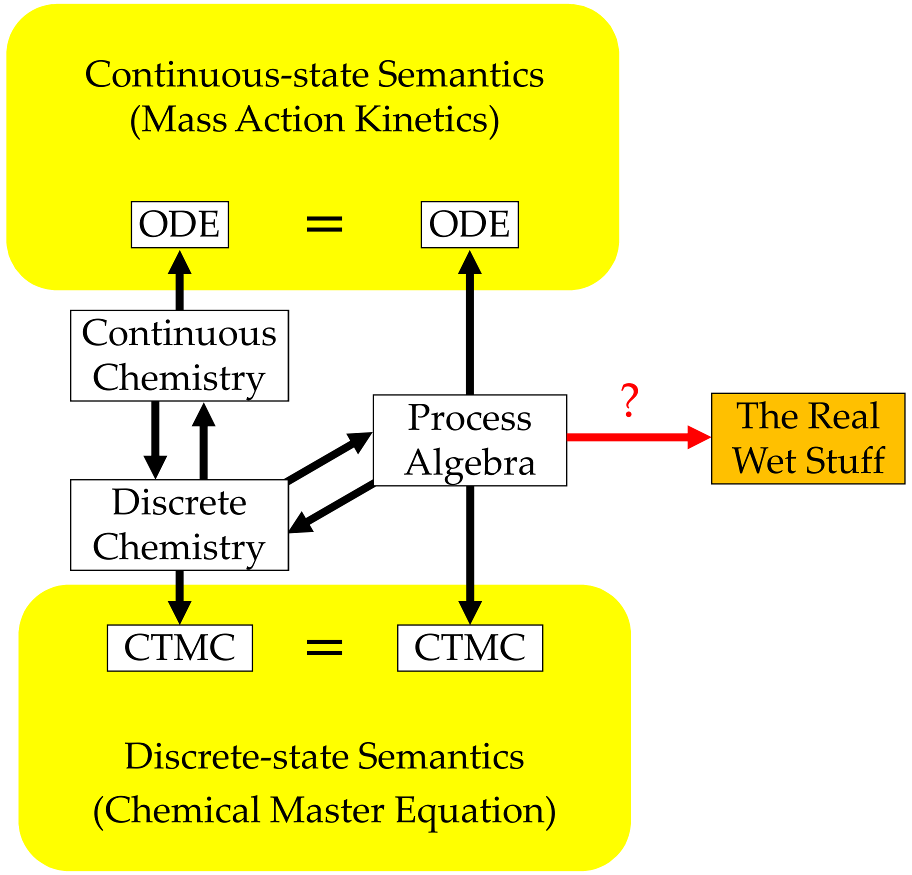
Part I Summary

Process Algebra is 'Bigger' and 'More Compact' than finite chemistry



Part II: From Automata to Molecules

Motivation

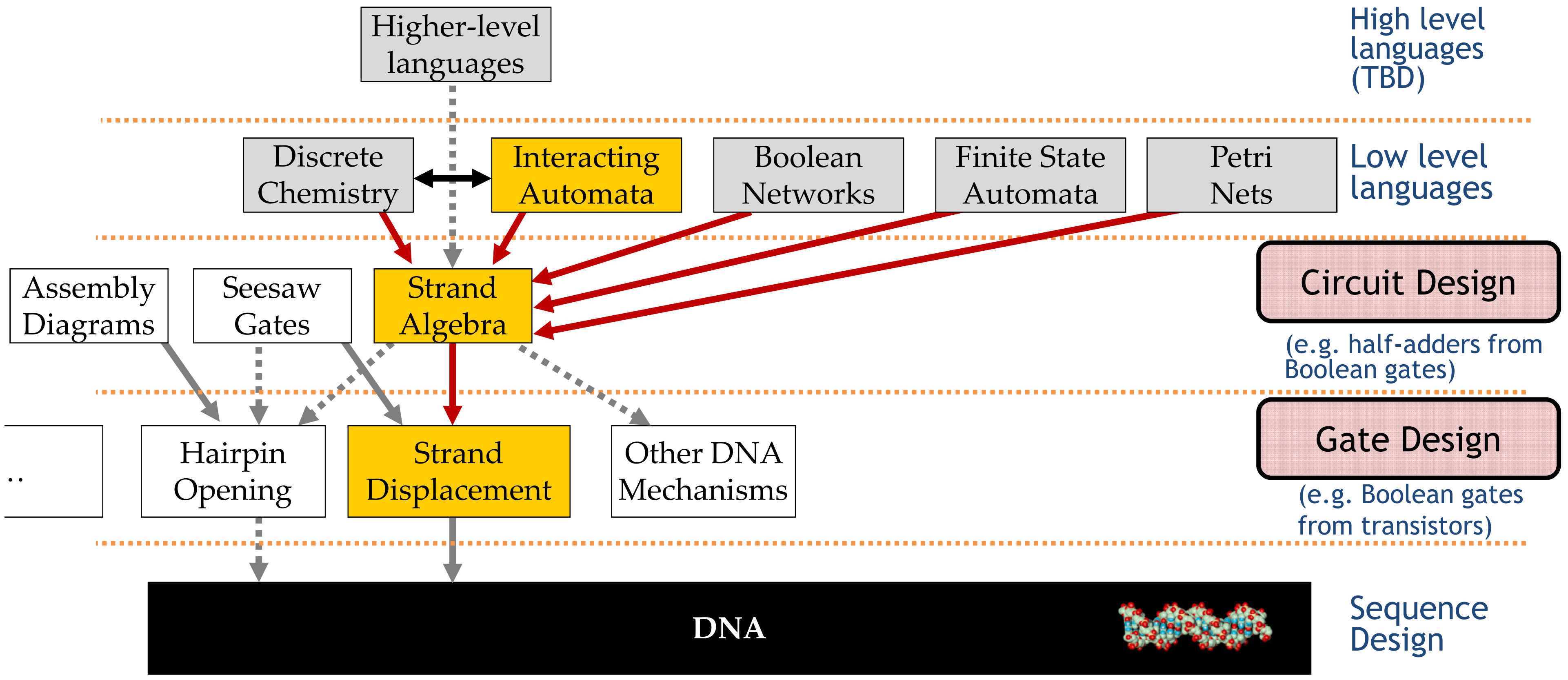


How do we implement an arbitrary process?

Chemistry does not necessarily help
(how do we then implement the chemical species?)

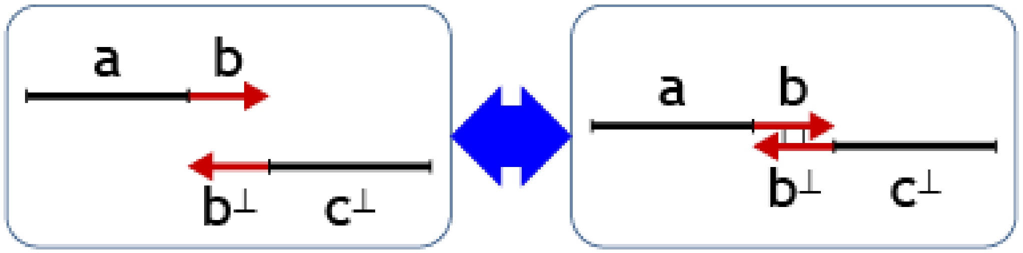
DNA Compilation

Separating Circuit Design from Gate Design



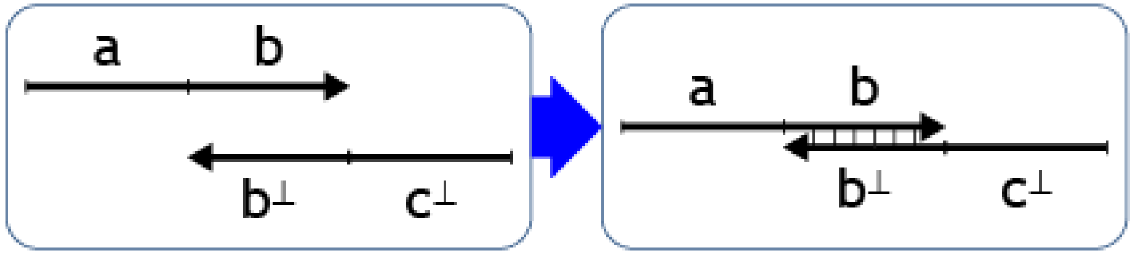
Gate Elements: Short and Long DNA Segments

Short (red) segments



Reversible Binding

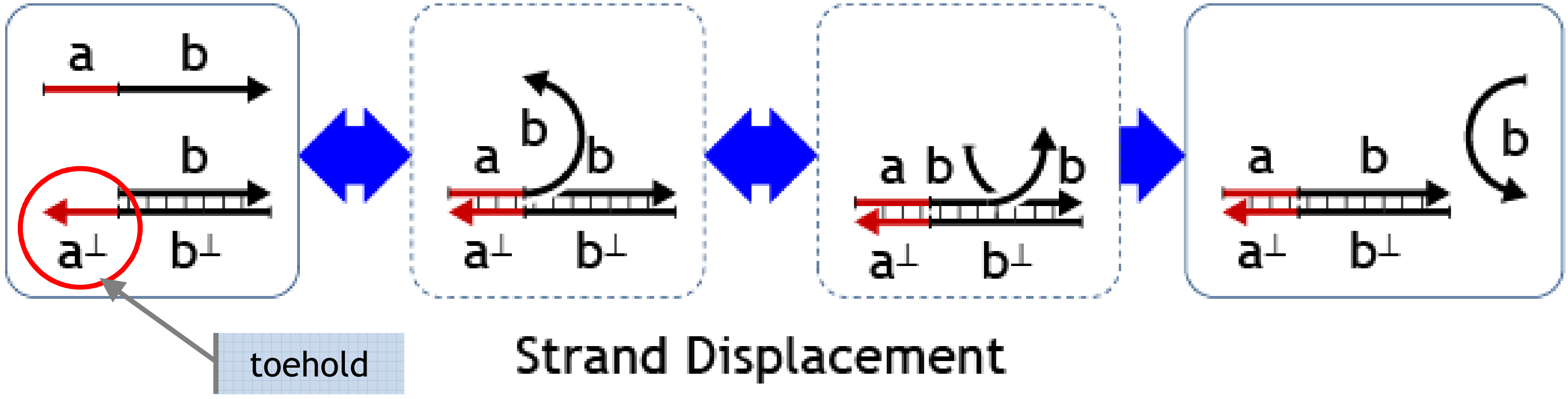
Long (black) segments



Irreversible Binding

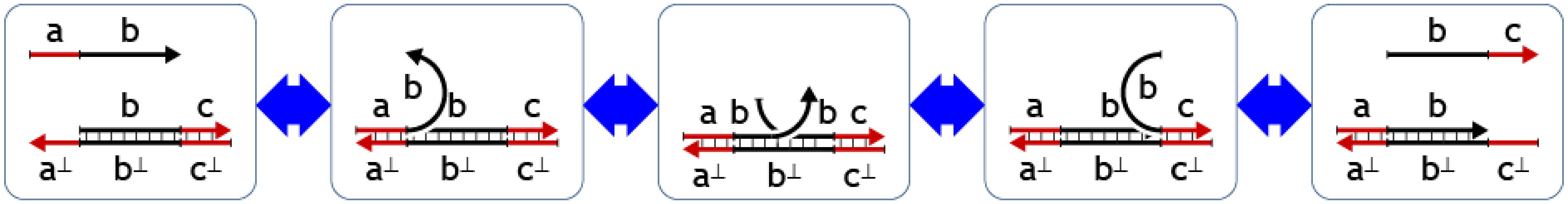
Gate Elements: Basic Mechanisms

Irreversible



Strand Displacement

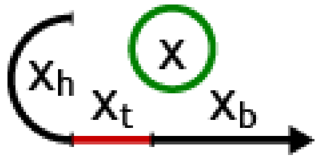
Reversible



Toehold Exchange

Gate Elements: Signals and Gates

- Signals "x" are single-stranded and 'positive'

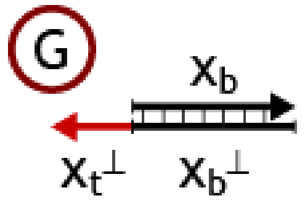


x_h = history
x_t = toehold
x_b = binding

x_t, x_b = signal identity for x

- This 3-segment signal representation is original to this work, it is based on the 4-segment signals of D. Soloveichik, G. Seelig, E. Winfree. Proc. DNA14, but leads to simpler and more regular gate structures

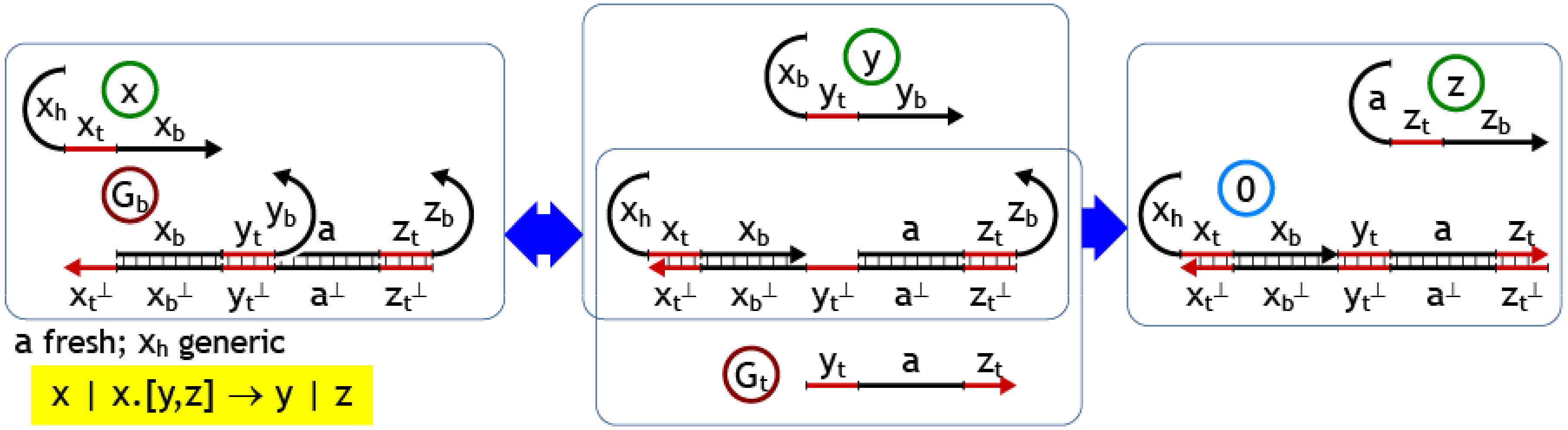
- Gate backbones are double-stranded, except for 'negative' toeholds.



- Separation of strands and gates helps the DNA realization, as one can use 3-letter alphabets (ATC/ATG) for each strand, minimizing secondary structure and entanglement.

Circuit Elements: x.[y,z] Fork Gate

- A Fork signal-processing gate takes a signal x and produces two signals y,z according to the reaction $x \mid x.[y,z] \rightarrow y \mid z$



x
x_h x_t x_b
G_b
x_b y_t y_b a z_t z_b
$x_t^\perp$ $x_b^\perp$ $y_t^\perp$ $a^\perp$ $z_t^\perp$
a fresh; x_h generic
$x \mid x.[y,z] \rightarrow y \mid z$
y
x_b y_t y_b
x_h x_t x_b a z_t z_b
G_t
y_t a z_t
z
a z_t z_b
0
x_h x_t x_b y_t a z_t

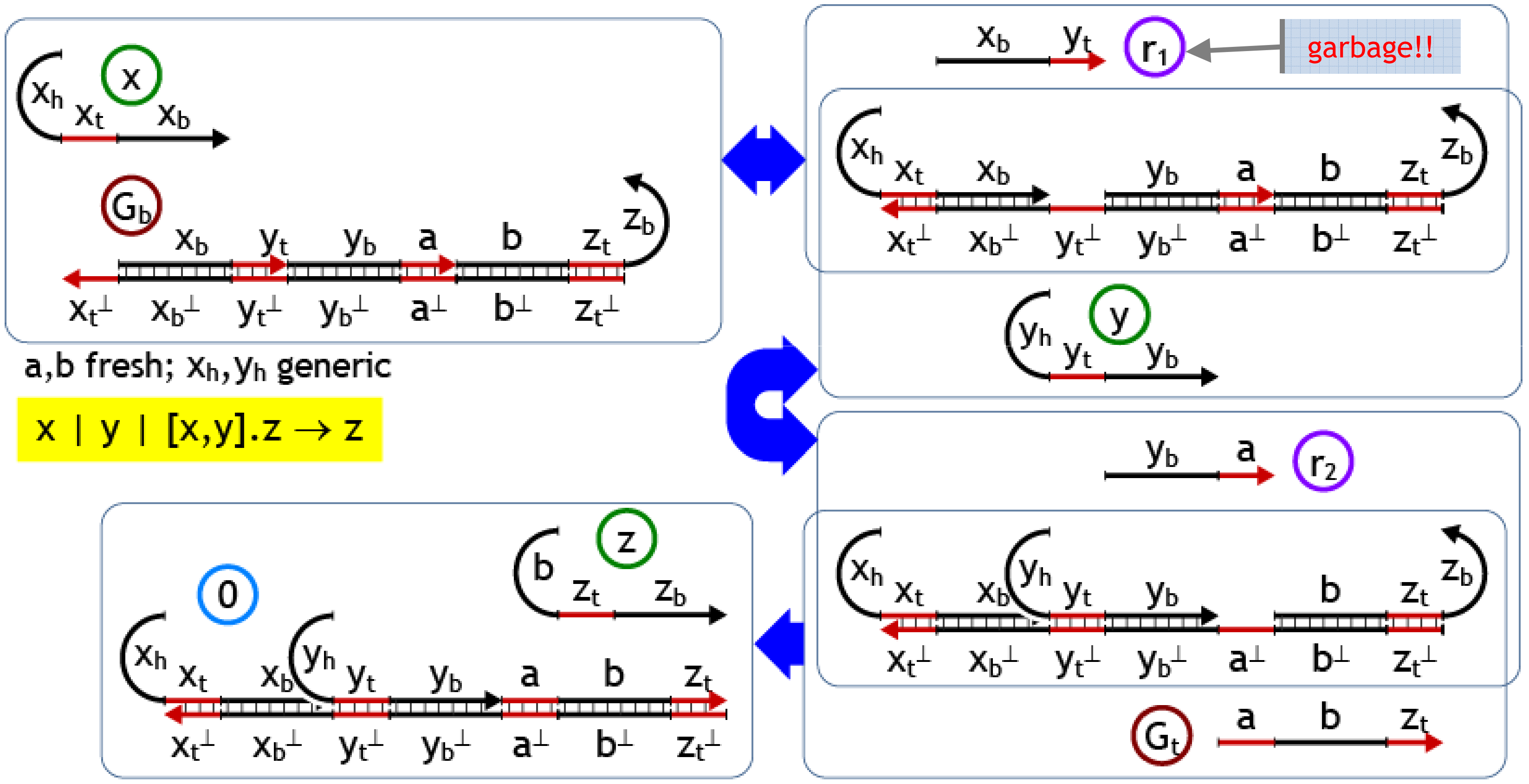
G_b,G_t (gate backbone and trigger) form the gate.

Any history segment that is not determined by the gate structure is said to be ‘generic’ (can be anything).

Any gate segment that is not a non-history segment of an input or output signal is taken to be ‘fresh’ (globally unique for the gate), to avoid possible interferences.

Circuit Elements: [x,y].z Join Gate (function)

- A Join signal-processing gate takes *both* signals x,y and produces a signal z according to the reaction $x \mid y \mid [x,y].z \rightarrow z$

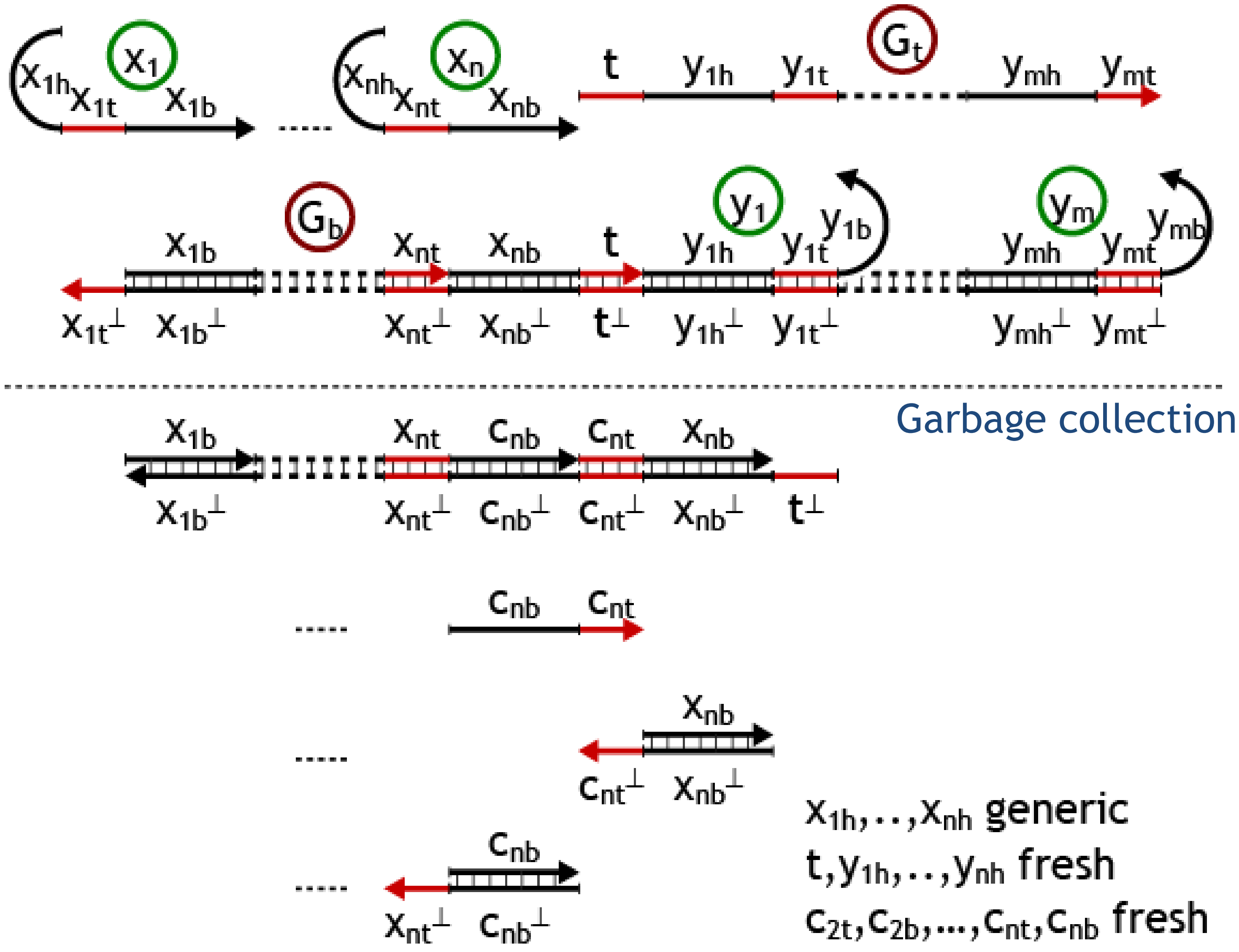


x
x_h x_t x_b
G_b
x_b y_t y_b a b z_t z_b
$x_t^\perp$ $x_b^\perp$ $y_t^\perp$ $y_b^\perp$ $a^\perp$ $b^\perp$ $z_t^\perp$
a,b fresh; x_h,y_h generic
$x \mid y \mid [x,y].z \rightarrow z$
x_b y_t r_1
garbage!!
y
y_h y_t y_b
y_b a r_2
G_t a b z_t
0
z
b z_t z_b

The garbage r_1 and r_2 must be collected (*after* the gate has fired) to avoid accumulation. This can be achieved by a similar scheme taking r_1,r_2 as input signals.

$[x_1,..,x_n].[y_1,..,y_m]$ General Join/Fork Gate

$x_1 \mid .. \mid x_n \mid [x_1,..,x_n].[y_1,..,y_m] \rightarrow y_1 \mid .. \mid y_m$



x1h x1t x1 x1b
xnh xnt xn xnb
t y1h y1t Gt ymh ymt
x1b Gb xnt xnb t y1 y1h y1t y1b ym ymh ymt ymb
x1t⊥ x1b⊥ xnt⊥ xnb⊥ t⊥ y1h⊥ y1t⊥ ymh⊥ ymt⊥
x1b xnt cnb cnt xnb
x1b⊥ xnt⊥ cnb⊥ cnt⊥ xnb⊥ t⊥
cnb cnt
xnb
cnt⊥ xnb⊥
cnb
xnt⊥ cnb⊥

Garbage collection

$x_{1h},..,x_{nh}$ generic

$t,y_{1h},..,y_{nh}$ fresh

$c_{2t},c_{2b},...,c_{nt},c_{nb}$ fresh

Strand Algebra

$$P \;::=\; x \;\vdots\; [x_1,..,x_n].[y_1,..,y_m] \;\vdots\; 0 \;\vdots\; P|P \;\vdots\; P^* \qquad n \geq 1,\ m \geq 0$$

x	is a *signal*
$[x_1,..,x_n].[y_1,..,y_m]$	is a *gate*
0	is an *inert solution*
$P|P$	is *parallel composition* of signals and gates
P^*	is a *population* (multiset) of signals and gates

Reaction Rule

$$x_1 \mid .. \mid x_n \mid [x_1,..,x_n].[y_1,..,y_m] \rightarrow y_1 \mid .. \mid y_m$$

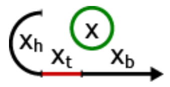
Auxiliary rules (axioms of diluted well-mixed solutions)

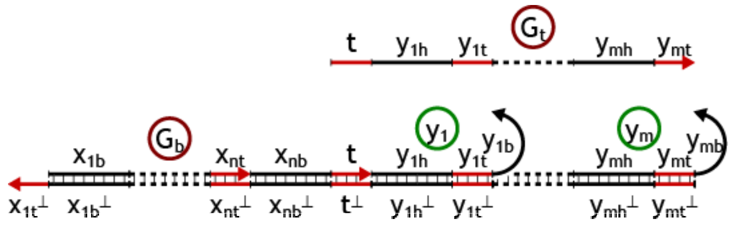
$P \rightarrow P' \;\Rightarrow\; P \mid P'' \rightarrow P' \mid P''$	Dilution
$P \equiv P_1,\ P_1 \rightarrow P_2,\ P_2 \equiv P' \;\Rightarrow\; P \rightarrow P'$	Well Mixing

Where $\equiv$ is a congruence relation (syntactical 'chemical mixing')
with $P^* \equiv P \mid P^*$ for unbounded populations.

Compiling Strand Algebra to DNA

P ::= x : $[x_1,..,x_n].[y_1,..,y_m]$: 0 : P|P : P* $n \geq 1,\ m \geq 0$

- compile(x) = 

- compile($[x_1,..,x_n].[y_1,..,y_m]$) = 

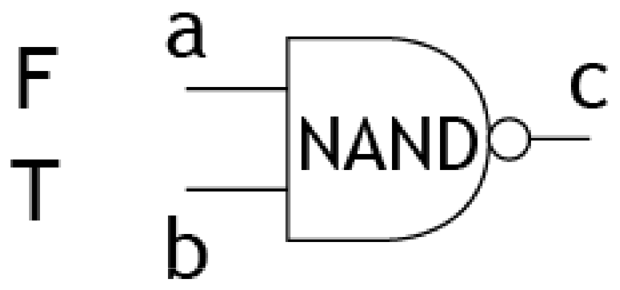
- compile(0) = empty solution

- compile(P | P') = mix(compile(P), compile(P'))

- compile(P*) = population(compile(P))

Boolean Networks

Boolean Networks to Strand Algebra



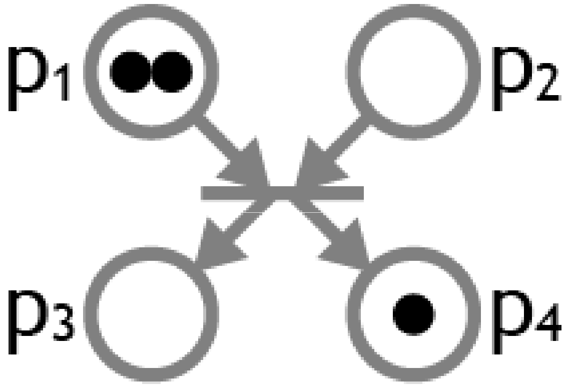
$$
\begin{array}{l}
([a_F, b_F].c_T)^* \mid \\
([a_F, b_T].c_T)^* \mid \\
([a_T, b_F].c_T)^* \mid \\
([a_T, b_T].c_F)^* \mid \\
a_F \mid b_T
\end{array}
$$

This encoding is *compositional*, and can encode *any* Boolean network:
- multi-stage networks can be assembled (combinatorial logic)
- network loops are allowed (sequential logic)

Petri Nets

Petri Nets to Strand Algebra

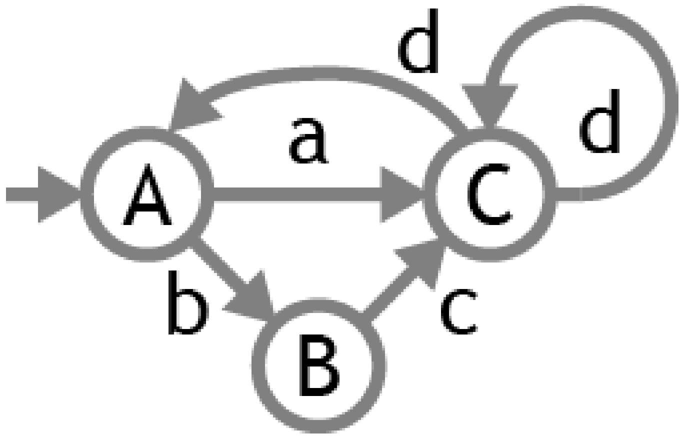
Transitions as Gates
Place markings as Signals



$([p_1,p_2].[p_3,p_4])^* \mid$
$p_1 \mid p_1 \mid p_4$

Finite State Automata

FSA to Strand Algebra



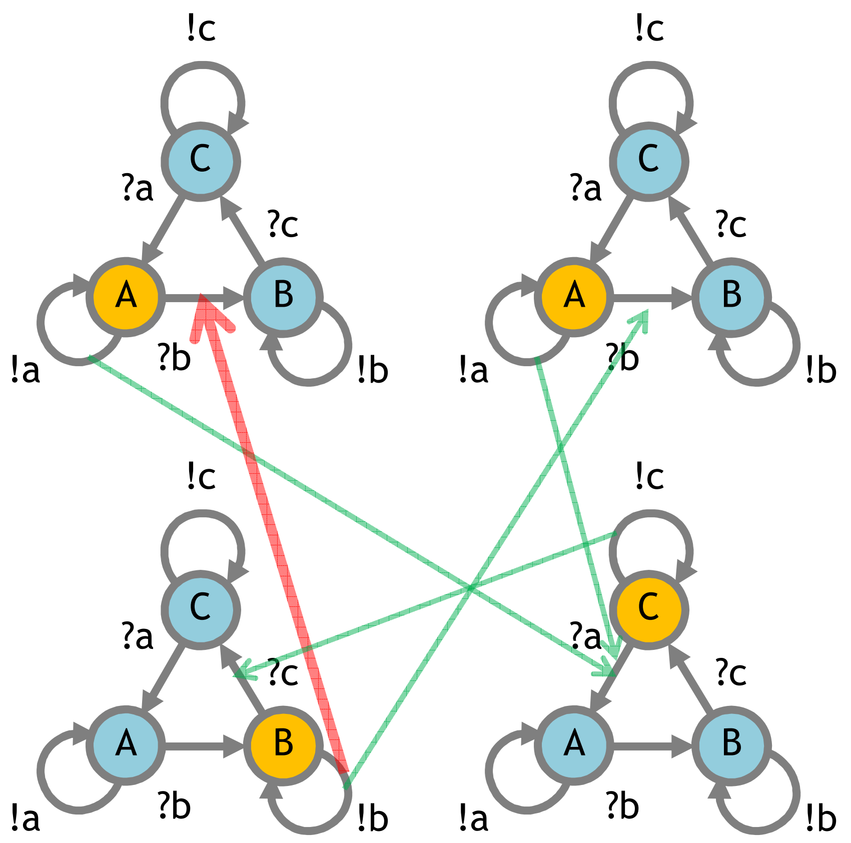
$$
\begin{aligned}
&([A,a].[C,\tau])^* \mid \\
&([A,b].[B,\tau])^* \mid \\
&([B,c].[C,\tau])^* \mid \\
&([C,d].[C,\tau])^* \mid \\
&([C,d].[A,\tau])^* \mid \\
&A \mid \tau
\end{aligned}
$$

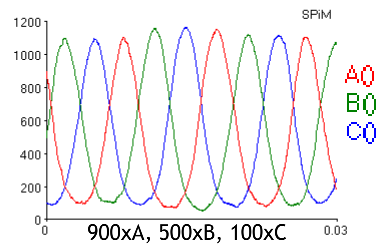
Input strings

a,b,c,d

$$
\begin{aligned}
&\tau\ .[a,\sigma_1] \mid \\
&[\sigma_1,\tau].[b,\sigma_2] \mid \\
&[\sigma_2,\tau].[c,\sigma_3] \mid \\
&[\sigma_3,\tau].\ d
\end{aligned}
$$

Interacting Automata

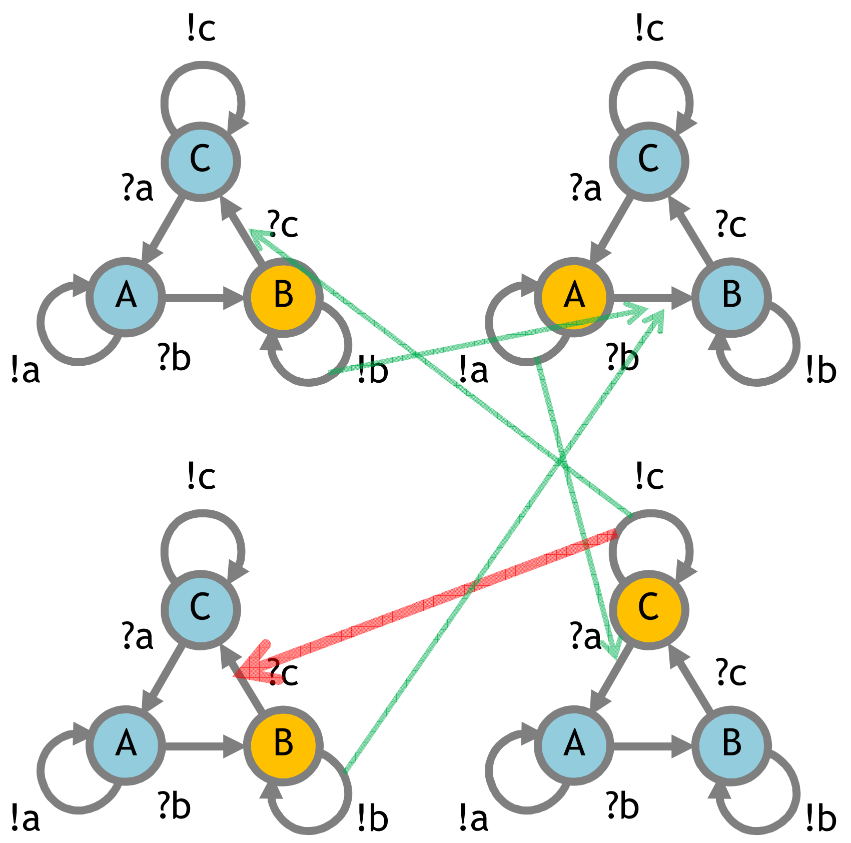


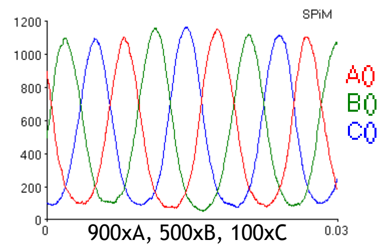


([A,B].[B,B])* |
([B,C].[C,C])* |
([C,A].[A,A])* |
A | A | B | C

This is a uniform population of identical automata,
but heterogeneous populations of interacting automata can be similarly handled.

Interacting Automata

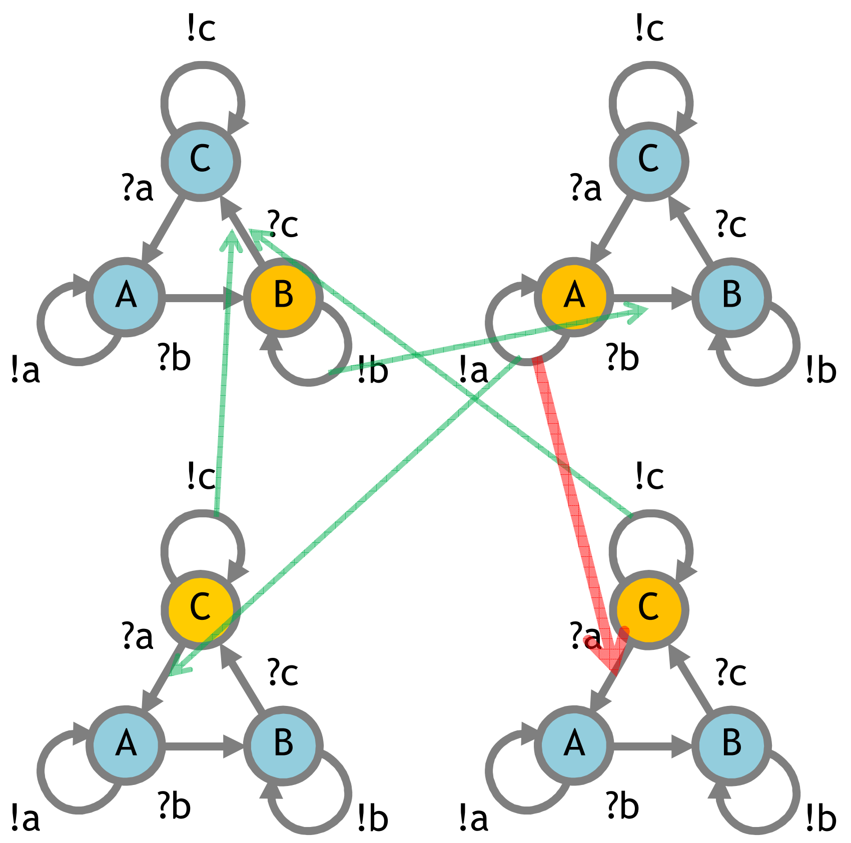


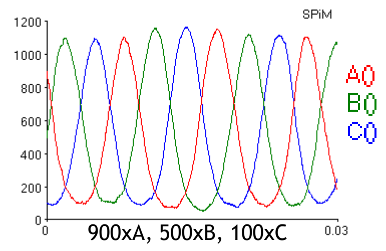


([A,B].[B,B])* |
([B,C].[C,C])* |
([C,A].[A,A])* |
A | B | B | C

This is a uniform population of identical automata,
but heterogeneous populations of interacting automata can be similarly handled.

Interacting Automata

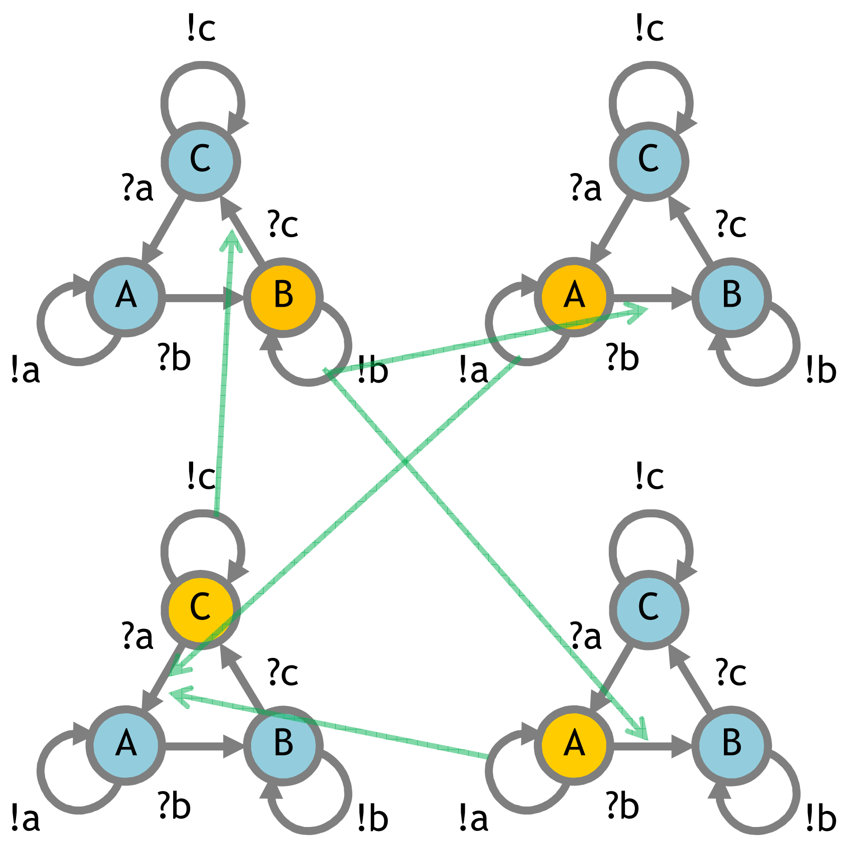


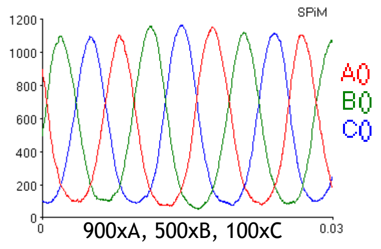


([A,B].[B,B])* |
([B,C].[C,C])* |
([C,A].[A,A])* |
A | B | C | C

This is a uniform population of identical automata,
but heterogeneous populations of interacting automata can be similarly handled.

Interacting Automata





([A,B].[B,B])* |
([B,C].[C,C])* |
([C,A].[A,A])* |
A | A | B | C

This is a uniform population of identical automata,
but heterogeneous populations of interacting automata can be similarly handled.

Conclusions

Conclusion

- History of computing
  - Is pointing towards increasing amounts of concurrency and heterogeneity in man-made systems.
  - Modern computer hardware is going (by necessity) multi-core.
  - Programming such systems is still a major challenge.

- Natural history
  - Massively concurrent and heterogeneous computation.
  - We do not really yet understand how concurrency works there (e.g. in gene networks, neural networks).

- Future molecular computing
  - Functional input-output devices? Individual automata?
  - Or reactive systems?