Artificial Biochemistry

Luca Cardelli

Microsoft Research

Abstract

Chemical and biochemical systems are presented as collectives of interacting stochastic automata: each automaton represents a molecule that undergoes state transitions. This framework constitutes an *artificial biochemistry*, where automata interact by the equivalent of the law of mass action. We analyze several example systems and networks, both by stochastic simulation and by ordinary differential equations.

1 Stochastic Automata Collectives

This paper is an empirical investigation of an *artificial biochemistry* obtained by the interactions of stochastic automata. The study of such artificial frameworks has been advocated before [2]; we explore a modern version based on a theory of concurrent processes that obeys the equivalent of the law of mass action. Foundations for this work have been investigated elsewhere [1]; here we aim to give a self-contained and accessible presentation of the framework, and to explore by means of examples the richness of "emergent" and unexpected behavior that can be represented by combinations of simple building blocks.

By a *collective* we mean a *large set* of *interacting*, *finite state* automata. This is not quite the situation we have in classical automata theory, because we are interested in the behavior of a large set of automata acting together. It is also not quite the situation with cellular automata, because our automata are interacting, but not necessarily on a regular grid. It is also not quite the situation in process algebra, because again we are interested in the behavior of collectives, not of individuals. Similar frameworks have been investigated under the headings of collectives [12], sometimes including stochasticity [6].

By *stochastic* we mean that automata interactions have rates. Stochastic rates induce a quantitative semantics for the behavior of collectives. Collective behavior cannot be considered quite discrete, because it can be the result of hundreds or thousands individual contributions. But it is not quite continuous either, because of the possibility of non-trivial stochastic effects. And it is also not hybrid: there is no switching between discrete and continuous regimes.

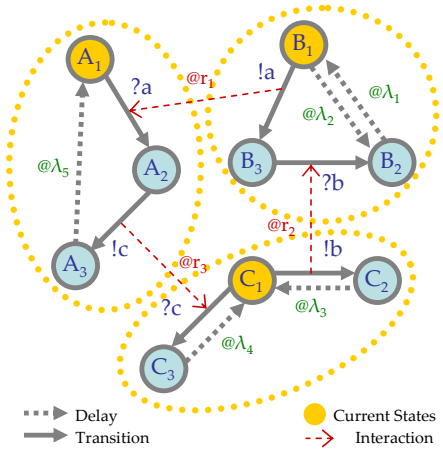

Figure 1 Interacting automata

Stochastic collectives are inspired by biochemical systems, which are large sets of interacting molecules/proteins, whose stochasticity ultimately derives from Brownian motion. An underlying as-

sumption, here, is that proteins can be regarded as *finite state* components that are subject to automata-like *transitions* between well-defined states. While certainly not accurate at the atomic level, this assumption is corroborated by the fact that much of the knowledge being accumulated in Systems Biology is described as state transition diagrams [5].

2 Interacting Automata

Therefore, we focus on the notion of stochastic interacting automata and their collective behavior. Figure 1 shows a typical situation. We have three separate automata species A, B, C (enclosed in yellow dotted envelopes), each with three possible states (circles) and with a *current* state: A_1, B_1, C_1, respectively. Thick gray arrows (solid or dashed) denote transitions between states of the same automaton; thin red dashed arrows denote interactions between separate automata. In general, systems consist of populations of automata, e.g. 100×A, 200×B and 300×C. Each automaton in a population, e.g. A, can have current state A_1, A_2, or A_3 at any given time.

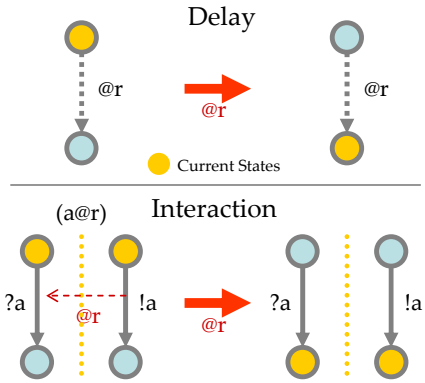

Figure 2 Automata reactions

There are two possible kinds of *reactions* that cause automata to change state (Figure 2). From the current state, an automaton can spontaneously execute a *delay* (dashed gray arrow). Or, it can jointly execute an *interaction* with another automaton (solid gray arrow). In an interaction, each of the two automata executes in a complementary way either an *input* (?) or an *output* (!) on a common *channel* (a channel is an abstract way to represent an interaction surface or mechanism). An actual interaction happens only if both automata are in a current state such that the interaction is enabled, and then both automata change state simultaneously.

Each reaction fires at (@) a rate r; these rates determines (stochastically) the probability of selecting one out of many possible reactions, and also the time spent between successive reactions.

The system of three automata in Figure 1, for example, could go through the following state changes: $A_1,B_1,C_1 \rightarrow A_2,B_3,C_1 \rightarrow A_2,B_2,C_2 \rightarrow A_2,B_2,C_1 \rightarrow A_3,B_2,C_3 \rightarrow A_1,B_2,C_3 \rightarrow A_1,B_2,C_1 \rightarrow A_1,B_1,C_1$.

2.1 Groupies and Celebrities

In the rest of this section we explore a little zoo of simple but surprising automata collectives, before beginning a more systematic study in Section 3. We set all our reaction rates to 1.0, since we are more interested in the effects of automata structure on behavior, than the effects of rate changes.

The automaton in Figure 3 has two possible states, A and B. A single automaton can perform no reaction, because all its reactions are interactions with other automata. Suppose that we have two such automata in state A; they each offer !a and ?a, hence they can interact on channel a, so that one moves to state B and the other one moves back to state A (either one, since there are two possible reactions A+A→A+B and A+A→B+A). If we have two automata in state B, one will similarly move to state A. If we have one in state A an one in state B, then no interactions are possible and the system is stable.

We call such automata *celebrities* because they aim to different from each other. How will a random population of celebrities behave? The stochastic simulation in Figure 3 [9] shows that a 50/50 equilibrium is reached and maintained. Moreover, the system is live: individual automata keep changing state.

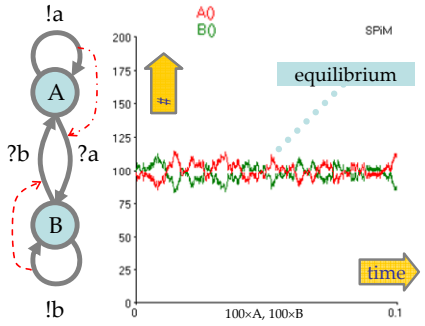

Figure 3 Celebrity automata

The interactions between celebrities are (misleadingly) indicated in Figure 3 by a thin red dashed line on the same automaton; remember that an automaton can never interact with itself, hence this notation always refers to interactions between distinct automata in a population of similar automata. In any case, the thin red dashed lines are just redun-

dant graphics: all the possible interactions can be read out from the !a/?a labels on transitions.

Let us now consider a different two-state automaton shown in Figure 4. Again, a single automaton can do nothing. Two automata in state A are stable since they both offer !a and ?b, and no interactions are possible. Similarly for two automata in state B. If we have one automaton in state A and one in state B, then they offer !a and ?a, so they can interact on channel a and both move to state A. They also offer ?b and !b, so they can (nondeterministically) interact on channel b and both move to state B.

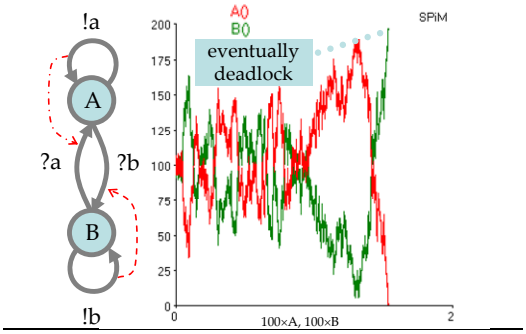

Figure 4 Groupie automata

We call such automata *groupies* because they aim to be similar. How will a random population of groupies behave? In Figure 4 we start with 50% A and 50% B: the systems evolves through a bounded random walk, and the outcome is uncertain till the very end. Eventually, though, the groupies end up forming a single homogeneous group of all A or all B, and the system is then dead: no automaton can change state any further.

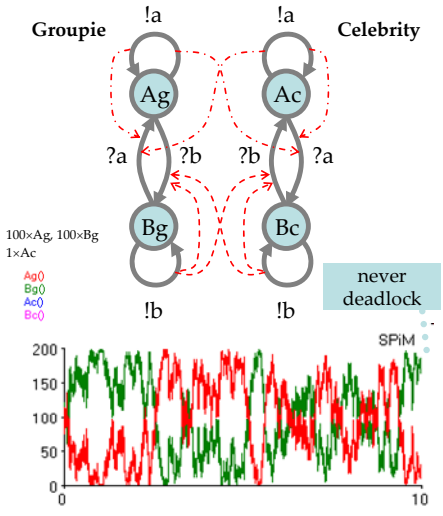

Figure 5 Both together

Populations of groupies and populations of celebrities have radically different behavior. What will happen if we mix them? It is sufficient to mix a small number of celebrities (1 is enough) with an arbitrarily large number of groupies, to achieve another radical change in system behavior. As shown in Figure 5, the groupies can still occasionally agree to become, e.g., all A. But then a celebrity moves to state B to differentiate itself from them, and that breaks the deadlock by causing at least one groupie to emulate the celebrity and move to state B. Hence, the whole system now evolves as a bounded random walk with no stable state. An infinitesimal number of celebrities has transformed the groupie behavior form a system that always eventually deadlocks, to a system that never deadlocks. We can replace celebrities with simpler *doping* automata (Figure 6) that have the same effect of destabilizing groupie collectives.

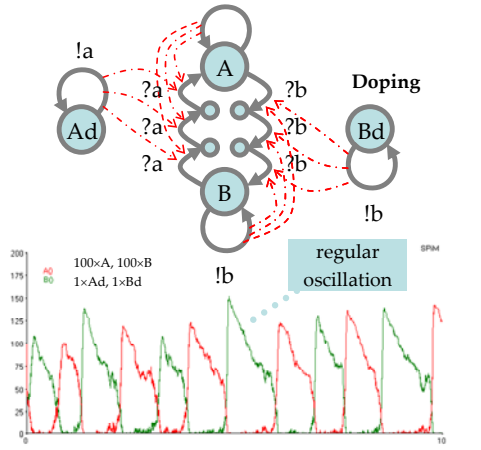

Figure 6 Hysteric groupies

We now change the structure of the groupie automaton by introducing intermediate states on the transitions between A and B (Figure 6), while still keeping all reactions rates at 1.0. Now, each groupie in state A must find three groupies in state B to be persuaded to change to state B. Once started the transition from A to B is irreversible; hence, some hysteresis (history-dependence) is introduced in the transition. The intermediate states produce a striking change in behavior: from complete randomness to regular oscillations. (Using one intermediate state instead of two yields a less regular oscillation.)

The oscillations are stochastic both in height and in width, and occasionally one may observe some miss-steps. But the transformation in behavior, obtained by changes in the structure of individual automata, is certainly remarkable (and largely independently on rate values). Moreover, this oscillator is critically dependent on an infinitesimal amount of

doping (i.e. and infinitesimal amount of stochastic noise); without it, it typically stops on its first cycle.

The morale from these examples is that the collective behavior of even the simplest interactive automata can be rich and surprising. Macroscopic behavior "emerges" from the structure of the components, and even an infinitesimal number of components can have macroscopic effects. The question then arises, how can we relate the macroscopic behavior to the microscopic structure?

3 The Chemistry of Automata

3.1 First Order Reactions

As we have seen, an automaton in state A can spontaneously move to state A′ at a specified rate r, by a stochastic delay. In a population of such automata, each transition decrements the number of automata in state A, and increments the number of automata in state A′. This can be written also as a chemical reaction $A \rightarrow^{r} A'$, with first-order rate law -r[A], where [A] is the number of automata in state A as a function of time. The rate of change in the number of A (assuming A′≠A) is the derivative of [A], written $[A]^{\bullet} = -r[A]$. The speed of the transition is thus an exponential decay at rate r, $Exp(r)(t) = re^{-rt}$.

Figure 7 First order reactions

A sequence of exponential decays Exp(r) produces an Erlang distribution Erl(r,k), as seen in Figure 8. Initially, we have N=10000 automata in state S_1. The occupation of the initial state S_1 is an exponential decay N·Exp(r) = N·Erl(r,1); the occupation of the intermediate states S_i is N·Erl(r,i); and the occupation of the final state is the cumulative distribution of Erl(r,10).

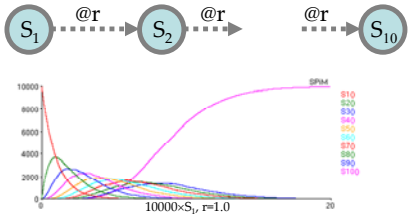

Figure 8 Sequence of delays

The shape of an exponential distribution is independent of the initial quantity (e.g., the half-life is constant). In general, for first order reactions, the time course of the reactions is independent of the scaling of the initial quantities. For example, if we start with 10 times as many automata in Figure 8, and we scale down the vertical axis by a factor of 10, we obtain the same plot up to time 20. Meaning that the "speed of the systems" is the same as before, and since there are 10 time more reactions in the same time, the "execution rate" is 10 times higher.

3.2 Second Order Reactions

Two automata can interact to perform a joint transition on a common channel, each changing its current state. The interaction is synchronous and complementary: one automaton in current state A performs an input $?a_{(r)}$ and moves to state A′; the other automaton in state B performs an output $!a_{(r)}$ and moves to state B′.

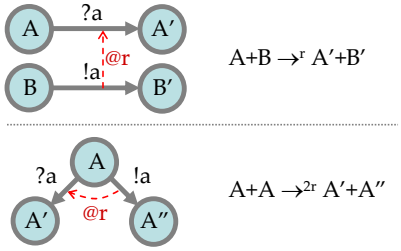

Figure 9 Second order reactions

This interaction can be written as a chemical reaction $A+B \rightarrow^{r} A'+B'$, where r is the fixed rate assigned to the interaction channel. The rate law, given by the law of mass action, is -r[A][B], and the rates (assuming A,B,A′,B′ all distinct states) are $[A]^{\bullet} = [B]^{\bullet} = -r[A][B]$, because each automaton in the population of current states [A] can interact with each automaton in the population of current states [B].

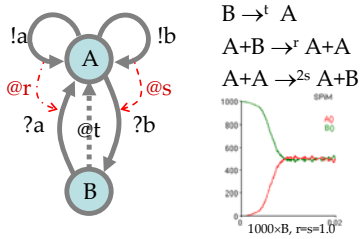

Figure 10 All 3 reactions in 1 automaton

A different situation arises, though, if the interaction happens within a single population, e.g., when state A offers both an input ?a to transition to state A′ and an output !a to transition to state A″. Then, every automaton in state A can interact with every *other* automaton in state A in *two* symmetric ways; hence, the product interactions are between [A] and [A]-1 automata, and the rate r must be doubled. The chemical reaction is $A+A \rightarrow^{2r} A'+A''$, whose rate law, based on the number of possible symmetric

collisions between particles at base rate 2r, is –(2r)[A]([A]-1)/2 = –r[A]([A]-1). The rate for [A] (assuming A'≠A≠A") is $[A]^{\bullet}$ = –2r[A]([A]-1), since 2 A are lost each time.

In Figure 10 we show an automaton that exhibits a first order reaction and one of each kind of second order reactions. Its collective behavior is determined by the corresponding chemical reactions.

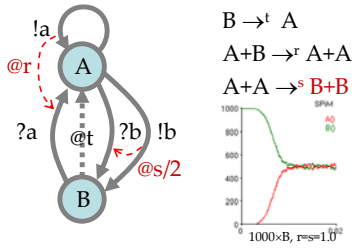

Figure 11 Same behavior

To compare automata behavior we must in general go beyond the chemical reactions, and compute the ODEs (in the standard chemical way) to compare the rates of state occupations. For example, the automaton in Figure 11 has a different pattern of interactions and rates, different chemical reactions, but the same ODEs as the one in Figure 10. In both cases, $[A]^{\bullet}$ = -$[B]^{\bullet}$ = t[B] + r[A][B] - s[A]([A]-1), but note that the b rate in Figure 11 set to s/2.

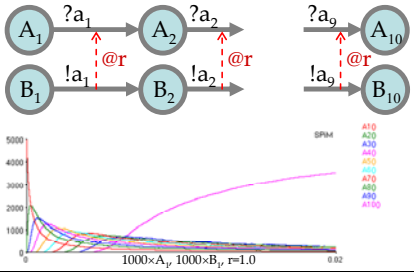

Figure 12 Sequence of interactions

The time course of second-order reactions decreases linearly with the scaling up of the initial quantities. For example, if we start with 10 times as many automata as in Figure 12, and we scale down the vertical axis by a factor of 10, and we scale up the time axis by a factor of 10, we obtain the same plot up to time 0.002. Meaning that the "speed of the system" is 10 times faster than before, and since there are also 10 times more reactions, the "execution rate" is 100 times higher. Moreover, the system in Figure 12, judging by the final output level, is about 1000 times faster than the one in Figure 8, in reaching 50% of input level (100 times faster in reaching 90% of input level), even though it has the same base rates and the same number of automata.

3.3 Zero Order Reactions

First order reactions have a law of the form r[A], and second order reactions a law of the form r[A][B]. Zero order reactions are those with a law of the form r, meaning that the "execution rate" is constant, and hence the "speed of a system" gets slower when more ingredients are added. Zero order reactions are not built into chemistry, but can be implemented (up to an approximation) by chemical means. The main biochemical methods of obtaining zero order reactions is a special case of enzyme kinetics, when enzymes are saturated.

Dealing with enzyme kinetics would bring us outside of the realm of simple automata (see Section 4). Instead, here we discuss a close analog of enzyme kinetics that exhibits zero-order behavior and can be represented within the automata framework described so far. We will make precise how this is a close analog of enzymes, and in fact, with a few assumptions, it can be used to model enzyme kinetics in a simpler way.

Consider the system of Figure 13. Here E is the (pseudo-) enzyme, S is the substrate being transformed with the help of the enzyme, and P is the product resulting from the transformation. The state ES represents an enzyme that is "not available" because it is busy transforming some S into some P. This system exhibits zero order kinetics, as can be seen from the plot. If we start with lots of S and a little E, the rate of production of P is constant (independent of the instantaneous quantity of S). That happens, of course, because E becomes maximally busy, and effectively processes S sequentially. Even if we add more enzyme (up to a point) it will normally be found in the ES state: to obtain the zero-order behavior it is not necessary to have a single E, just that most E be normally busy. All our rates are 1.0 as usual, but note that E→ES happens fast, proportionally to [E][S], while ES→E happens slowly, proportionally to [ES].

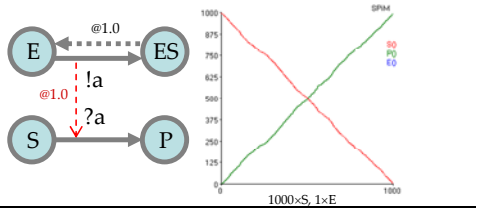

Figure 13 Zero order reactions

To make the connection to enzyme kinetics precise, we now mimic the standard derivation of Michaelis-Menten kinetics from chemical reactions. The reactions for the system in Figure 13 are:

$$E + S \rightarrow^{r} ES + P$$
$$ES \rightarrow^{s} E$$

The corresponding ODEs are:

$$[E]^{\bullet} = s[ES] - r[E][S]$$
$$[ES]^{\bullet} = r[E][S] - s[ES]$$
$$[S]^{\bullet} = - r[E][S]$$
$$[P]^{\bullet} = r[E][S]$$

We call $[E_0] = [E]+[ES]$ the total amount of enzyme, either free or busy; that is, the number of enzyme automata. We now assume that, in normal operation, the enzyme is in equilibrium, and in particular $[ES]^{\bullet}=0$. This implies that $[ES] = r[E][S]/s$. Set:

$$K_m = s/r$$
$$V_{max} = s[E_0]$$

Hence $[ES] = [E][S]/K_m$, and $[ES] = ([E_0]-[ES])[S]/K_m$, and from that we obtain $[ES] = [E_0]([S]/(K_m+[S]))$. From the $[ES]^{\bullet}=0$ assumption we also have $[P]^{\bullet} = s[ES]$, and substituting [ES] yields:

$$[P]^{\bullet} = V_{max}[S]/(K_m+[S]).$$

Noticeably, if we have $K_m << [S]$, then $[P]^{\bullet} \approx V_{max}$; that is, we are in the zero-order regime, with constant growth rate $V_{max} = s[E_0]$. For the system of Figure 13 we have $K_m = 1$, $[S]_0 = 1000$, and $V_{max} = 1$, and hence $[P]^{\bullet} \approx 1$, as shown in the simulation.

The chemical reactions from Figure 13 are significantly different from the standard enzymatic reactions. Still, the expressions for K_m, V_{max}, and $[P]^{\bullet}$ turn out to be the same as in Michaelis-Menten kinetics (and not just for the zero-order case), whenever the dissociation rate is negligible, that is, for good enzymes.

3.4 Ultrasensitivity

Zero-order kinetics can be used, rather paradoxically, to obtain sudden non-linearity or switching behavior. Let us first compare the result of directly competing enzymes in zero-order and second-order kinetics. We now depict a pair of states E,ES (as in Figure 13) as a single state E with a solid/dashed arrow representing the transition through the hidden state ES (Figure 14).

At the top of Figure 14, in zero-order regime, an initial quantity of F is competing against a linearly growing quantity of E. The circuit is essentially computing [F]-[E], so the E quantity is neutralized until it exceeds the F quantity. (The plots Eb and Fb are for the bound states of E and F.) At the bottom, in second-order regime, with no bound states, the result of the competition is quite different.

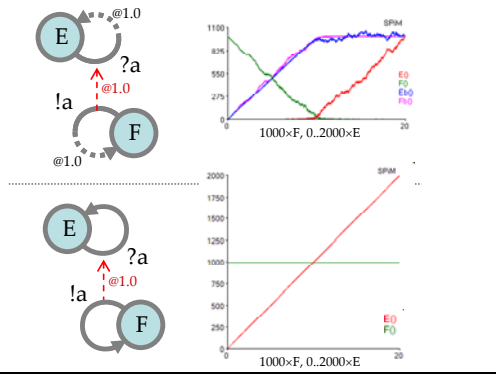

Figure 14 Subtraction (top)

On that basis, we now reproduce the peculiar phenomenon of hypersensitivity in zero-order regime [7], confirming that our simplified kinetics reproduces effects of enzyme kinetics. In a hypersensitivity situation, a minor switch in relative enzyme quantities, creates a much amplified and sudden switch in two other quantities.

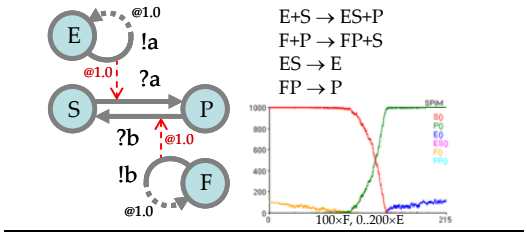

Figure 15 Ultrasensitivity

In Figure 15, we start with a fixed amount (=100) of enzyme F, which is holding the S-P equilibrium in the S state (=1000), and we let E grow from zero. As E grows, we do not initially observe much free E, but the level of free F decreases, indicating it is getting "harder" for F to maintain the S equilibrium against E. Eventually the level of free F drops to zero, at which point we see a sudden switch of the S-P equilibrium, and then we observe the level of free E growing. Hence, in this case, a switch in the levels of E vs. F controls a factor of 10 bigger switch in the levels of S vs. P. If P is itself an enzyme, it can then cause an even bigger and even more sudden switch of an even larger equilibrium.

3.5 Positive Feedback Transitions

Another way to obtain sharp transitions is by positive feedback with second-order reactions. In Figure 16, the more B's there are, the faster the A's are transformed into B's. Note that at least one B is needed to bootstrap the process.

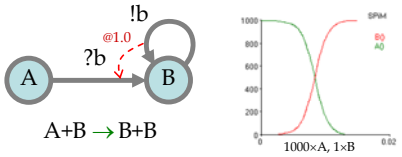

Figure 16 Positive feedback transition

When linking two such transitions, in Figure 17, we obtain a symmetrical bell shape with kinetics $[B]^{\bullet} = [B]([A]-[C])$. We provide 1000×A, 1×B and 1×C; note that there is a very small chance that the B's be drained by the C's before the wave can accumulate in B, therefore stalling it.

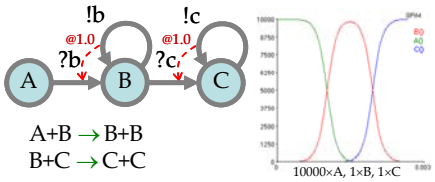

Figure 17 Bell shape

Linking several such transitions produces a soliton-like wave; but again this requires that each stage be able to bootstrap itself. This can be difficult to arrange because each stage is being pulled and emptied by the next one. One solution is to add a delay from each stage to the next, so that an incoming wave leaks into the next stage and starts the bootstrap (which is equivalent to doping the transition, as in Section 2). This works rather well, stochastically, but corresponds deterministically to a slightly dampened propagation due to the leaking.

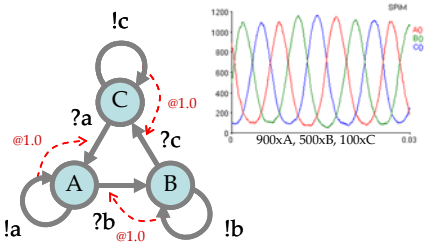

Figure 18 Oscillator

Linking three positive feedback transitions in a loop produces a stochastic oscillator (Figure 18); the oscillatory behavior can be verified also by extracting the ODEs form the chemical reactions. A sustained oscillation can be obtained by starting with all states non-zero; the oscillation can then survive as long as no state A,B,C goes to zero, and this can be arranged with very high probability. If any state touches zero, there is nothing to pull on the next wave, and the oscillation stops. Adding delays from each state into the next one, to prevent that possibility, produces a slightly dampened oscillator which stochastically goes through instabilities.

An interesting variation is a two-stage positive feedback loop, Figure 19, where the drop of state A is delayed and the growth of state B is steeper. Joining two such transitions (Figure 20) produces a shapes that approximates a rectangular wave as we increase the cardinality of A.

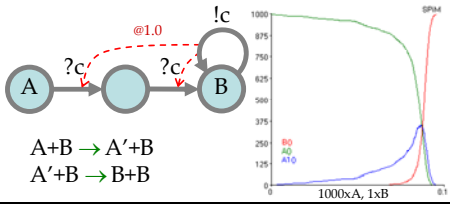

Figure 19 Positive two-stage feedback

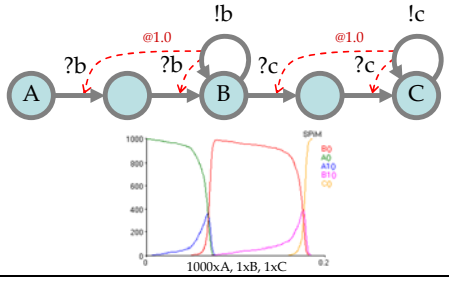

Figure 20 Square shape

Linking three such transitions in a loop produces again an oscillator. However, this time it is critical to add doping because each state regularly drops to zero and needs to be repopulated to start the next propagation.

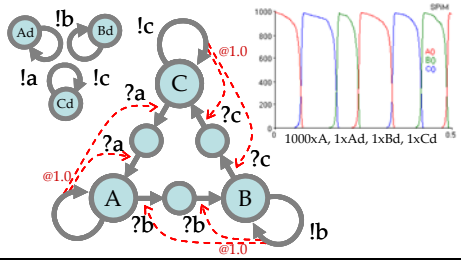

Figure 21 Hysteric 3-way groupies

3.6 Excitation Cascades

We now consider cascades where one enzyme activates another enzyme. A typical situation is shown in Figure 22 (again, all rates are 1.0), where a low constant level of first-stage aHi results in a maximum level of third-stage cHi, and where, characteristically, the third stage raises with a sigmoidal shape, and faster than the second-stage level of bHi.

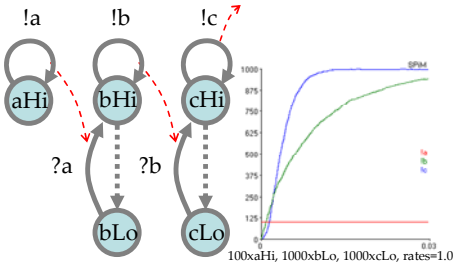

Figure 22 Second-order cascade

This network can be considered as the skeleton of MAPK cascades, which similarly function as amplifiers with three stages of activation, but are more complex in structure and detail [4].

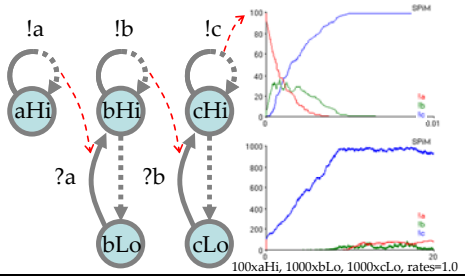

Figure 23 Zero-order cascade

The resulting amplification behavior, however, is non-obvious, as can be seen by comparison with the zero-order activation cascade in Figure 23; the only difference is in the zero-order kinetics of the enzymes obtained by introducing a delay of 1.0 after each output interaction. Within the same time scale as before, the level of cHi raises quickly to the (lower) level of aHi, until aHi is all bound. On a much longer time scale, cHi then grows linearly to maximum. Linear amplification in cascades has been attributed to negative feedback [11], but apparently can be obtained also by zero-order kinetics. Of course, the behavior in Figure 22 is the limit of that in Figure 23, as we decrease the zero-order delay.

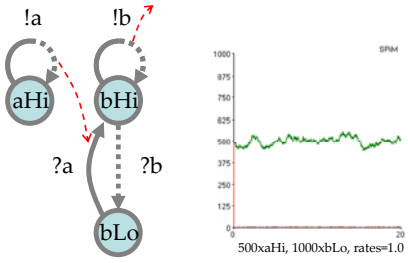

Figure 24 Zero-order transduction

A single stage of a second-order cascade works like the bHi level shown in Figure 22 (since no bHi is actually consumed by the next stage), that is, as an amplifier. A single stage of the zero-order cascade, however, work quite differently, as a signal replicator. In Figure 24, a given level of aHi (=500), as long as it is lower than the reservoir of bLo (=1000), is transformed into an equal level of bHi (=500). (If aHi exceeds bLo, then bHi:=bLo and aHi:=aHi-bLo.) However, the two-stage cascade in Figure 23 does not work like two signal replicators in series! This seems to happen because the bHi are not available for degradation to bLo while bound by interaction with the next stage, cLo, and hence can accumulate.

Real MAPK cascades are actually based on double activation, as shown in Figure 25, where the sigmoid output is more pronounced and delayed than in Figure 22.

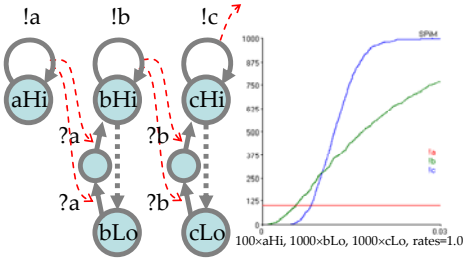

Figure 25 Second-order double cascade

And once again, the zero order regime brings surprises: the cascade in Figure 26 works in reverse, as a signal attenuator, where a high level of aHi produces a low stable level of cHi. This is because each stage of such a cascade is actually a signal level divider, with the signal being distributed among the three states of the stage.

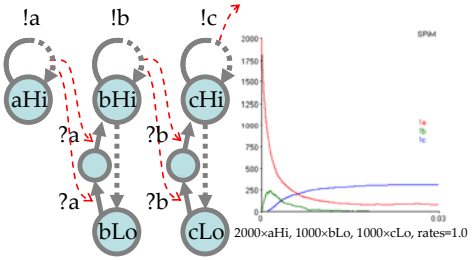

Figure 26 Zero-order double cascade

3.7 Boolean Inverters

Automata with distinguished "low" states and "high" states can be used to represent respectively Boolean *false* and *true*. We begin by investigating automata collectives that implement Boolean inverters. The most obvious inverter, Inv(a,b) with input ?a and output !b, is shown in Figure 27: its natural state is high because of the spontaneous decay from low to high. The high state sustains (by a self loop) an output signal (b) that can be used as input to further gates. A high input signal (a) pulls the high state down to low, therefore inverting the input.

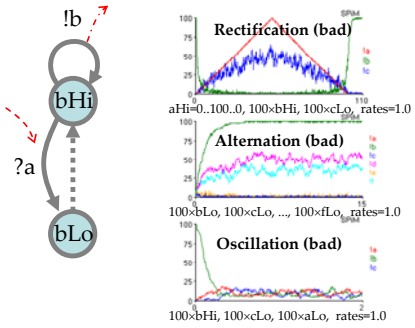

Figure 27 Simple inverter

Populations of such automata are tested in three ways:

(1) Rectification. With populations 100×Inv(a,b) + 100×Inv(b,c), a triangularly-shaped test signal (?a) is provided that ramps up from 0 to 100 and then back down to 0. We can see that the inverter is very responsive, quickly switching to low !b output and then quickly switching back to high !b output when the input is removed. But the !c output is neither a faithful reproduction nor a Boolean rectification of the ?a input.

(2) Alternation. A chain 100×Inv(a,b) + 100×Inv(b,c) + … + 100×Inv(e,f), leads to intermediate signals that are neither high nor low.

(3) Oscillation. A loop 100×Inv(a,b) + 100×Inv(b,c) + 100×Inv(c,a) fails to sustain a Boolean oscillation.

Therefore, we conclude that this inverter does not have good Boolean characteristics, possibly because it reacts strongly to a very small input levels, instead of switching on a substantial signal.

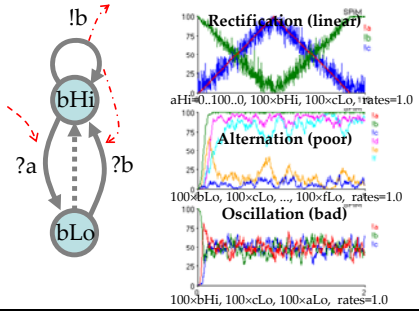

Figure 28 Feedback inverter

In an attempt to force a Boolean behavior, we add a positive feedback to the high state, so that (one might think) a higher input level would be required to force switching, hence improving the Boolean switching characteristics. The result is unexpected, but still interesting: a linear signal inverter. (We can deduce the linearity from the ODEs derived from the chemistry of this automaton: at steady state we have [bLo] = [aHi][bHi]/([bHi]+1) ≈ [aHi], hence [bHi] = max-[bLo] ≈ max-[aHi]). Such a linear inverter can be useful for inverting an analog signal, and also has decent Boolean alternation properties. But it does not oscillate.

A good Boolean inverter can be obtained, instead, by doubling the height of the simple inverter. This double height inverter gives perfect alternation, and good rectification (transforming a triangular input into a nearly rectangular output). However, it still fails to oscillate.

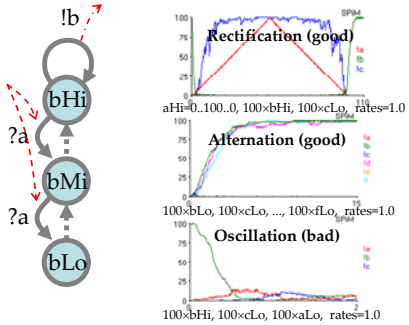

Figure 29 Double-height inverter

Finally, we combine the two techniques in a double-height feedback inverter. This has perfect rectification, transforming a triangular input into a sharp rectangle. It also has strong and quickly achieved alternation, and very regular oscillation.

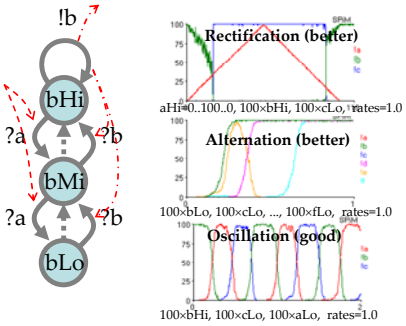

Figure 30 Double-height feedback inverter

In conclusion, it is possible to build good Boolean inverters and rectifiers. This is important because it lessens the requirements on other Boolean circuits: even if those circuits degrade signals, we can always use a rectifier to restore a proper Boolean signal. We examine some Boolean circuits next.

3.8 Boolean Circuits

We consider automata with low states and high states to represent respectively Boolean *true* and *false*. In general, to implement Boolean functions, we also need to use intermediate states and multiple high and low states.

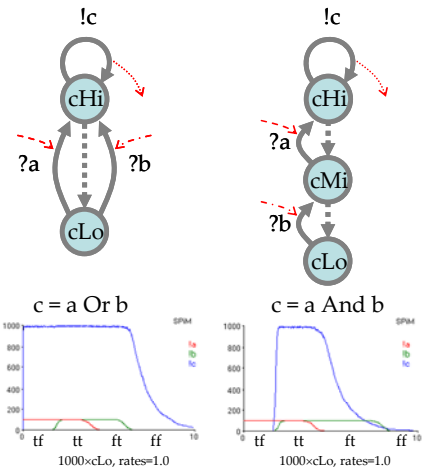

Figure 31 Or and And

Figure 31 shows the Boolean gate automata for "c = a Or b" and "c = a And b" The high states spontaneously relax to low states, and the low states are driven up by other automata providing inputs to the gates (not shown). A self-loop on the high states provides the output. In the plots, two input signals that partially overlap in time are used to test all four input combination; their high level is just 1/10 of max (where max is the number of gate automata).

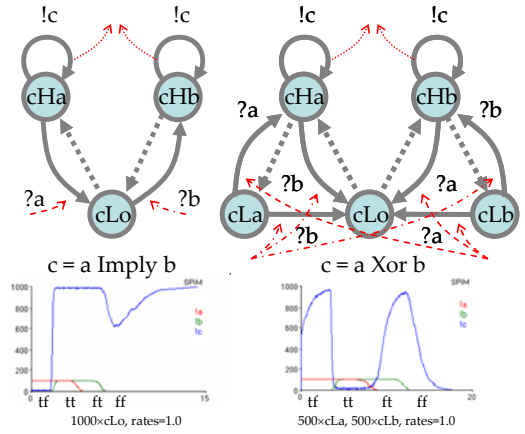

Figure 32 Imply and Xor

The chemical reactions for the Or gate are aHi+cLo→aHi+cHi, bHi+cLo→bHi+cHi, cHi→cLo. From those, we can derive the ODEs and set the derivatives to zero to analyze the steady state. With the automata constraint [cHi]+[cLo] = max, we obtain [cHi] = max([aHi]+[bHi])/([aHi]+[bHi]+1). That is, if the inputs are zero then [cHi] = 0. If instead, say, [aHi] is non-zero, then [cHi] = max[aHi]/([aHi]+1) so that for large [aHi] we have [cHi] ~ max.

Figure 32 shows automata for "c = a Imply b" and "c = a Xor b". In these automata we use two high states (both producing the same output) to respond to different inputs. The dip in the plot for Imply arises when many automata decay from the high state cHa to the high state cHb, through cLo, in a transition from *false* Imply *true* to *false* Imply *false*.

Analyzing the steady state behavior of Imply we obtain: output = [cHa] + [cHb] = max - max[aHi] / ([aHi][bHi] + [aHi] + 1) where max is the size of the collective. Hence, if [aHi]=0 we have output=max; if [aHi]≠0 and [bHi]=0 we have output~0; otherwise assume [aHi]~[bHi]~max, in which case we have output~max.

Although Xor can be constructed from a network of simpler gates, Figure 32 shows an Xor gate implemented as a single uniform collective.

4 The Biochemistry of Automata

4.1 Beyond simple automata

A characteristic feature of biochemistry, and of proteins in particular, is that biological molecules can stick to each other (forming *complexes*) while preserving their identity, and can later separate. This behavior can be represented by chemical reactions, but only by considering a complex as a brand new chemical species, thus losing the notion of molecular identity. Moreover, *polymers* are formed by the iterated complexation of similar molecules (*monomers*). Chemically this can be represented only by an unbounded number of different chemical species, one for each length of a polymer, which is obviously cumbersome.

In order to model the complexation features of biochemistry accurately and conveniently, we must move from individual automata to automata that form reversible complexes. Thus, we now consider *polyautomata*: automata that can *associate*, in addition to interacting as usual. *Association* represents the event of joining two specific automata together out of a population, and *dissociation* is the event that causes two specific associated automata to break free; both events result in state changes. Association does not prevent an automaton from performing normal interactions or other associations, but it prevents it from reassociating on the same interface, unless if first dissociates.

Association and dissociation can be encoded in π-calculus [8], by taking advantage of one of its most powerful features, resulting in flexible modeling of complexation and polymerization [10]. However, here we strive to remain within the confines of an automata-like framework, including diagrammatic descriptions.

4.2 Polyautomata

Polyautomata are automata with an association history, and with additional kinds of interactions that modify such history. The main formal difference from the automata of Section 2 is that the current state now carries with it a set S of *past associations*. An association is a pair $\langle \pi,k \rangle$ where π is an input action ?a or output action !a responsible for an association event, and k is a *unique* integer identifying an association event between two automata. We assume that a *fresh* k can be produced from, e.g., a global counter during the evolution of a collective; only two automata should have the same k in their associations at any given time. This unique k is used to guarantee that the *same* two automata that associated in the past will dissociate in the future.

There can be many ways of disassociating two automata after a given association, and association and disassociation events can have their own rates. Therefore, each channel is now attributed with a list of one or more rates: this is written $a@r_0,...,r_{n-1}$ for $n \geq 1$. We then say that arity(a)=n.

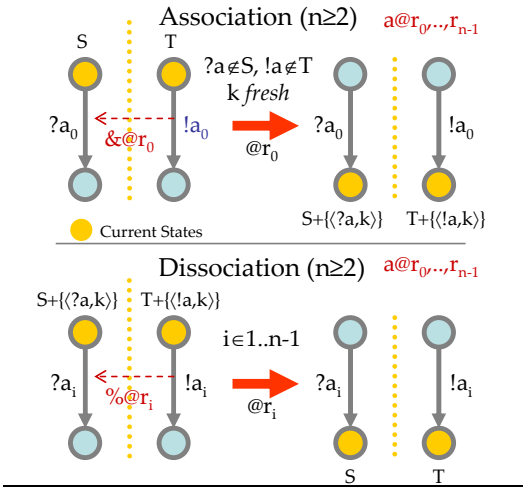

Figure 33 Rules of association

If arity(a)=1, then r_0 is called the *interaction rate*, and the old interaction rules from Figure 2 apply with $r_0 \equiv r$, with the understanding that the association sets are unaffected. If arity(a)≥2, then r_0 is the *association rate*, and $r_1,...,r_{n-1}$ are the *dissociation rates*. The association rules from Figure 33 then apply.

An association (Figure 33 top) on a channel *cannot* happen if an automaton has a similar past association on that channel, as recorded in the current state; that is, that particular "surface patch" of the automaton is currently occupied and cannot be reused. The tests ?a∉S (short for $\langle ?a,k \rangle \notin S$ for any k) and !a∉T, check for such conditions, where S and T are the sets of associations. If a new association is possible, then a fresh integer k is chosen and stored in the association sets after the transition. The transition labels are $?a_0$ and $!a_0$, indicating an association at rate r_0 on channel a. For emphasis, in examples we use the notation &?a and &!a for these labels, where & indicates association, omitting index 0.

Symmetrically, a dissociation (Figure 33 bottom) on a channel happens only if the two automata have a past association on that channel, as identified by the same k in their current states. If a dissociation is possible, the corresponding associations are removed from the association sets after the transition (+ here is disjoint union), enabling further associations. The transition labels are $?a_i$ and $!a_i$ with $i \in 1..n-1$, indicating a dissociation at rate r_i on channel a. For emphasis, in examples we use the notation $\%?a_i$ and $\%!a_i$ for these labels, where % indicates dissociation; if arity(a)=2 then we write simply %?a and %!a, omitting index 1.

4.3 Complexation

As an example of the association/dissociation notation, in Figure 34 we consider two automata that cyclically associate, moving to *bound* states A_b, B_b, and then dissociate, moving back to *free* states A_f, B_f. We also show the association sets under each state, although the number k can change at each iteration. The cartoon illustrates the mechanics of complexation, where complexation channels are depicted as complementary surfaces. Finally, the plot shows that for the chosen rates, the equilibrium is heavily biased towards the bound states.

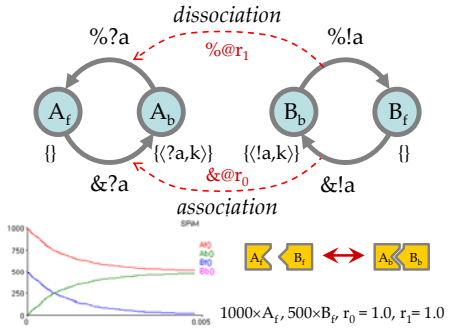

Figure 34 Complexation/decomplexation

The use of multiple dissociation rates is exemplified by enzymatic reactions, in Figure 35. From the bound state of enzyme (E_b) and substrate (S_b), two dissociations are possible at different rates, one of them producing product (P).

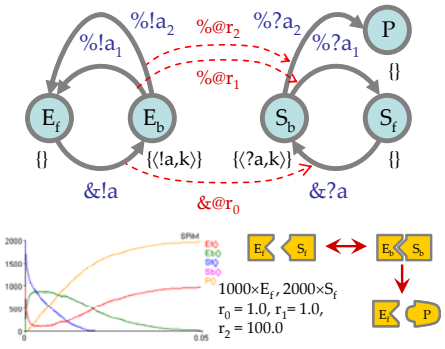

Figure 35 Enzymatic reactions

Homodimerization (Figure 36) is symmetric complexation: a monomer offers both an input and an output complexation on the same channel, meaning that it offers two complementary surfaces.

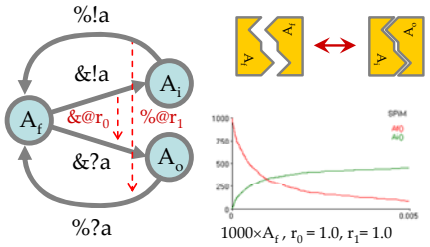

Figure 36 Homodimerization

The sequencing of states in this automaton guarantees that a complexation must be followed by a decomplexation, and that, for example, a monomer cannot bind to two other monomers over its two complementary surfaces. That situation leads to polymerization, as shown in the next section.

4.4 Polymerization

A *polymer* is obtained by the unbounded combination of *monomers* out of a finite set of monomer shapes. There are many forms of polymerization; here we consider just two basic ones.

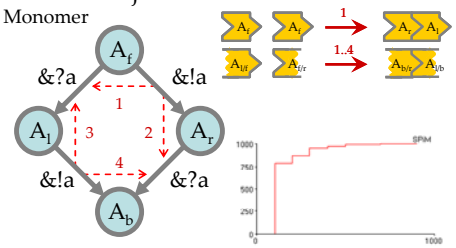

Figure 37 Bidirectional polymerization

In linear bidirectional polymerization, each monomer can join other monomers on one of two complementary surfaces, without further restrictions. Therefore, two polymers can also join in the same way, and a single polymer can form a loop (although a single monomer cannot). For simplicity, we do not allow these polymers to break apart.

In Figure 37 we show a monomer automata for this situation: it can be in one of four states: A_f (free), A_l (bound on the left), A_r bound on the right), and A_b (bound on both sides). The sequence of transitions is obviously from free, to bound on either side, to bound on both sides. There are four possible input/output associations between two monomers, indicated by the red dashed arrows in the figure. Number 1 is the association of two free monomers in state A_f: one becomes bound to the left (A_l) and the other bound to the right (A_r). Number 2 is the association of a free monomer with the leftmost monomer of a polymer (a monomer bound to the right): the free monomer becomes bound to the right and the leftmost monomer becomes bound on both sides (A_b). Number 3 is the symmetric situation of a free monomer binding to the right of a polymer. Number 4 is the leftmost monomer of a polymer binding to the rightmost monomer of another polymer (or possibly of the same polymer, forming a loop, as long as the two monomers are distinct).

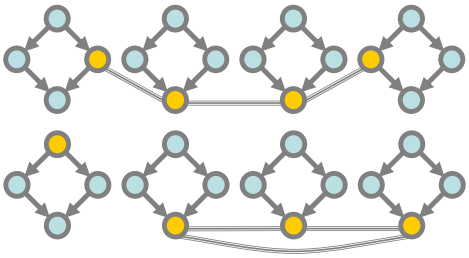

Figure 38 Automata polymers

The plot in Figure 37 shows the result of a fairly typical simulation run with 1000 monomers. When all the monomers are fully associated, we are left with a number of circular polymers: the plot is obtained by scanning the circular polymers after they stabilize. The horizontal axis is discrete and counts the number of such polymers (9 in this case). Each vertical step corresponds to the length of one of the circular polymers (polymers are picked at random for plotting: the vertical steps are not sorted by size). It is typical to find one very long polymer in the set (~800 in this case), and a small number of total polymers (<10).

We now consider a more constrained form of polymerization, inspired by the actin biopolymer, which can grow only at one end and shrink only at the other end. We have the same states as before, but the sequencing of transitions is now different.

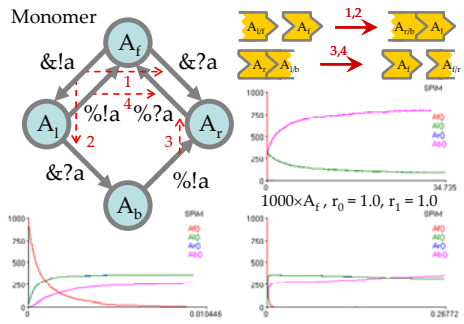

Figure 39 Actin-like polymerization

There are four possible input/output associations between two monomers, indicated by the red dashed arrows in the figure. Number 1 is the association of two free monomers in states A_f: one becomes bound to the left (A_l) and the other bound to the right (A_r). Number 4 is the breakup of a polymer made of just two monomers, one that is bound to the left and one that is bound to the right; they both return free. Number 2 is the association of a free monomer with the rightmost monomer of a polymer (a monomer bound to the left): the free monomer becomes bound to the left and the rightmost monomer becomes bound on both sides (A_b). Number 3 is the dissociation of a monomer bound to the right, from the leftmost monomer to its right which is bound on both sides; one becomes a free monomer and the other remains bound to the right. Note that loops cannot form here, because if we have a monomer bound to the left and one bound to the right (which could be the two ends of the same polymer), then there is no interaction that can make them bound on both sides.

The plots in Figure 39 show three views of the same simulation run with 1000 monomers, at times 0.01, 0.25, and 35 counterclockwise; all rates are 1.0. During an initial quick transient the number of A_b and of A_l=A_r temporarily stabilize, each approaching level 333 (with average polymer length 3). A_b crosses over at time 0.02 and then slowly grows until A_l=A_r=100 around time 35, meaning that the final number of polymers is ~100 with average length ~10.

Figure 40 shows a typical sequence of interactions among three monomers, with two associations followed by two dissociations. At each step we show the possible interactions by dashed red arrows connecting the enabled transitions. The triple lines indicate the complexation state, which is actually encoded in the association sets shown under the current states, by the shared k and j.

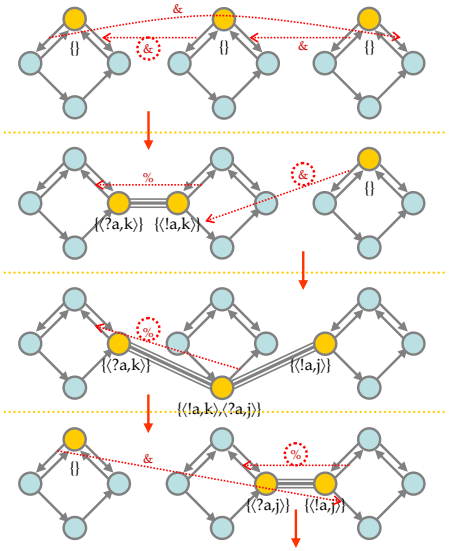

Figure 40 Typical monomer interactions

Note that in state A_b, the association set has the form {⟨!a,k⟩,⟨?a,j⟩}. This illustrates the need to store !a and ?a separately in the history: if we recorded only the channel, ⟨a,k⟩, then the second association for ⟨a,j⟩ would be prevented because the set would already contain the channel a. And if we modified the occurrence check to allow storing distinct pairs ⟨a,k⟩, ⟨a,j⟩, this would allow arbitrary reassociations on the same channel.

5 Conclusions

Despite ongoing conscious efforts, biochemistry is still lacking an adequate notation for describing large and complex biological models in a compositional, parameterizable, and scalable way [5]. In that respect, the role of *mathematical notation* (such as programming languages and process algebra) is fundamental in computing: it enables engineering and analysis techniques that are orthogonal to the ones available in *mathematical models* (such as set theory, calculus, and Markov chains). It is fair to say that adequate notation alone allows the maintainability of very large information processing systems, consisting of millions of lines of code, whose complexity is dwarfed only by biological systems. Operating systems are not written in differential equations, nor in set theory.

Still, it is always important to relate mathematical notation to mathematical models. We have used automata notation for exploring simple but intriguing biochemical systems, also aiming to demonstrate how easy it is to "play with" the notation to get insights into the systems. We have shown, by exam-

ple, how to relate the notation to stochastic behavior and to differential equations.

As a graphical notation, automata are compositional, but are neither parameterizable nor scalable [3]. However, they can be embedded in the richer framework of process algebras, which have such properties (although we have carefully avoided going down that path here). The pragmatics of the notation still needs to be worked out for the domain at hand. It has taken decades to develop adequate notations and analysis techniques for large software and hardware systems; we are just at the beginning of doing the same for biochemical systems, where the task will certainly be much harder.

References

[1] L. Cardelli: On process rate semantics. Available from http://lucacardelli.name. To appear.

[2] W. Fontana, L. W. Buss: The barrier of objects, from dynamical systems to bounded organization. In: Boundaries and Barriers, J. Casti and A. Karlqvist (eds.), pp. 56-116, Addison-Wesley, 1996

[3] D. Harel. Statecharts: a visual formalism for complex systems. Science of Computer Programming 8:231-274. North-Holland 1987.

[4] C-Y. F. Huang, J. E. Ferrell Jr.: Ultrasensitivity in the mitogen-activated protein cascade. Proc. Natl. Acad. Sci. USA, 93, 10078-10083. 1996.

[5] H. Kitano: A graphical notation for biochemical networks. BioSilico 1(5): 169-76. 2003.

[6] K. Lerman, A. Galstyan: Automatically modeling group behavior of simple agents. Agent Modeling Workshop, AAMAS-04, New York. 2004.

[7] M. H. Meinke, J. S. Bishops, R. D. Edstrom: Zero-order ultrasensitivity in the regulation of glycogen phosphorylase. Proc. Natl. Acad. Sci. USA, 83, 2865-2868, May 1986.

[8] Milner, R.: Communicating and Mobile Systems: The π-Calculus. Cambridge University Press, 1999.

[9] A. Phillips, L. Cardelli: A Correct Abstract Machine for the Stochastic Pi-calculus. Proc. BioConcur'04.

[10] Priami, C.; Regev, A.; Shapiro, E.; Silverman, W.: Application of a stochastic name-passing calculus to representation and simulation of molecular processes. Information Processing Letters 80, 25-31. 2001.

[11] H. M. Sauroa, B. N. Kholodenko: Quantitative analysis of signaling networks. Progress in Biophysics & Molecular Biology 86 5–43. 2004.

[12] K. Tumer, D. Wolpert: A survey of collectives. In Collectives and the Design of Complex Systems, 1-42. Springer, 200

Appendix: Simulation Scripts

The simulation are carried out in the SPiM stochastic π-calculus simulator [9].

Figure 3: Celebrity Automata

```
directive sample 0.1
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())
```

Figure 4: Groupie Automata

```
directive sample 2.0
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())
```

Figure 5: Both together

```
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())
```

Figure 6: Hysteric groupies

```
directive sample 10.0
directive plot A(); B()

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

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

let Ad() = !a; Ad()
and Bd() = !b; Bd()

run 100 of (A() | B())
run 1 of (Ad() | Bd())
```

Figure 8: Sequence of delays

```
directive sample 20.0
directive plot S1(); S2(); S3(); S4(); S5(); S6();
S7(); S8(); S9(); S10()

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; S10()
and S10() = ()

run 10000 of S1()
```

Figure 10: All 3 reactions in 1 automaton

```
directive sample 0.02
directive plot A(); B()

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

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

run 1000 of  B()
```

Figure 11: Same behavior

```
directive sample 0.002 10000
directive plot A(); B()

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

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

run 10000 of  B()
```

Figure 12: Sequence of interactions

```
directive sample 0.1
directive plot A1(); A2(); A3(); A4(); A5(); A6();
A7(); A8(); A9(); A10()

new a1@1.0:chan new a2@1.0:chan new a3@1.0:chan
new a4@1.0:chan new a5@1.0:chan new a6@1.0:chan
new a7@1.0:chan new a8@1.0:chan new a9@1.0:chan

let A1() = ?a1; A2()
and B1() = !a1; B2()
and A2() = ?a2; A3()
and B2() = !a2; B3()
and A3() = ?a3; A4()
and B3() = !a3; B4()
and A4() = ?a4; A5()
and B4() = !a4; B5()
and A5() = ?a5; A6()
and B5() = !a5; B6()
and A6() = ?a6; A7()
and B6() = !a6; B7()
and A7() = ?a7; A8()
and B7() = !a7; B8()
and A8() = ?a8; A9()
and B8() = !a8; B9()
and A9() = ?a9; A10()
and B9() = !a9; B10()
and A10() = ()
and B10() = ()

run 1000 of (A1() | B1())
```

Figure 13: Zero order reactions

```
directive sample 1000.0
directive plot S(); P(); E()

new a@1.0:chan()

let E() = !a; delay@1.0; E()
and S() = ?a; P()
and P() = ()

run (1 of  E() | 1000 of S())
```

Figure 14: Subtraction

```
directive sample 20.0 1000
directive plot E(); F(); Eb(); Fb()

new a@1.0:chan()

let E() = ?a; Eb()
and Eb() = delay@1.0; E()
and F() = !a; Fb()
and Fb() = delay@1.0; F()

let clock(t:float, tick:chan) =
  (* sends a tick every t time *)
  (val ti = t/100.0 val d = 1.0/ti
  (* by 100-step erlang timers *)
```

```
   let step(n:int) = if n<=0
     then !tick; clock(t,tick)
     else delay@d; step(n-1)
   run step(100))
let Sig(p:proc(), tick:chan) =
  (p() | ?tick; Sig(p,tick))
let raising(p:proc(), t:float) =
  (new tick:chan run (clock(t,tick) |
   Sig(p,tick)))

run 1000 of F()
run raising(E,0.01)
```

```
directive sample 20.0 1000
directive plot E(); F()

new a@1.0:chan()

let E() = ?a; E()
and F() = !a; F()

let raising(p:proc(), t:float) =
  ... see code for Figure 14

run 1000 of F()
run raising(E,0.01)
```

Figure 15: Ultrasensitivity

```
directive sample 215.0
directive plot S();P();E();ES();F();FP()

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

let S() = ?a; P()
and P() = ?b; S()

let E() = !a; delay@1.0; E()
and F() = !b; delay@1.0; F()

run 1000 of S()

let raising(p:proc(), t:float) =
  ... see code for Figure 14

run 100 of F()
run raising(E,1.0)
```

Figure 16: Positive feedback transition

```
directive sample 0.02 1000
directive plot B(); A()

val s=1.0

new b@s:chan
let A() = ?b; B()
and B() = !b;B()

run (1000 of A() | 1 of B())
```

Figure 17: Bell shape

```
directive sample 0.003 1000
directive plot B(); A(); C()

new b@1.0:chan new c@1.0:chan

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

run ((10000 of A()) | B() | C())
```

Figure 18: Bell shape

```
directive sample 0.03 1000
directive plot A(); B(); C()

new a@1.0:chan new b@1.0:chan new c@1.0:chan
let A() = do !a;A() or ?b; B()
and B() = do !b;B() or ?c; C()
and C() = do !c;C() or ?a; A()

run (900 of A() | 500 of B() | 100 of C())
```

Figure 19: Positive two-stage feedback

```
directive sample 0.07 1000
directive plot B(); A(); A1()

val s=1.0

new c@s:chan
let A() = ?c; A1()
and A1() = ?c; B()
and B() = !c;B()

run (1000 of A() | 1 of B())
```

Figure 20: Square shape

```
directive sample 0.2 1000
directive plot B(); A(); A1(); B1(); C()

new b@1.0:chan new c@1.0:chan

let A() = ?b; A1()
and A1() = ?b; B()
and B() = do !b;B() or ?c; B1()
and B1() = ?c; C()
and C() = !c;C()

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

Figure 21: Hysteric 3-way groupies

```
directive sample 0.5 1000
directive plot A(); B(); C()

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

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

let Ad() = !a; Ad()
and Bd() = !b; Bd()
and Cd() = !c; Cd()

run 1000 of A()
run 1 of (Ad() | Bd() | Cd())
```

Figure 22: Second-order cascade

```
directive sample 0.03
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Amp_hi(a:chan, b:chan) =
  do !b; Amp_hi(a,b) or delay@1.0; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_hi(a,b)

run 1000 of (Amp_lo(a,b) | Amp_lo(b,c))

let A() = !a; A()
run 100 of A()
```

Figure 23: Zero-order cascade

```
directive sample 0.01
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Amp_hi(a:chan, b:chan) =
  do !b; delay@1.0; Amp_hi(a,b)
  or delay@1.0; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_hi(a,b)

run 1000 of (Amp_lo(a,b) | Amp_lo(b,c))

let A() = !a; delay@1.0; A()
run 100 of A()
```

```
directive sample 20.0
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Amp_hi(a:chan, b:chan) =
  do !b; delay@1.0; Amp_hi(a,b)
```

```
  or delay@1.0; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_hi(a,b)

run 1000 of (Amp_lo(a,b) | Amp_lo(b,c))

let A() = !a; delay@1.0; A()
run 100 of A()
```

Figure 24: Zero-order transduction

```
directive sample 20.0
directive plot !a; !b

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

let Amp_hi(a:chan, b:chan) =
  do !b; delay@1.0; Amp_hi(a,b)
  or delay@1.0; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_hi(a,b)

run 1000 of Amp_lo(a,b)

let A() = !a; delay@1.0; A()
run 500 of A()
```

Figure 25: Second-order double cascade

```
directive sample 0.03
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Amp_hi(a:chan, b:chan) =
  do !b; Amp_hi(a,b) or delay@1.0; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; ?a; Amp_hi(a,b)

run 1000 of (Amp_lo(a,b) | Amp_lo(b,c))

let A() = !a; A()
run 100 of A()
```

Figure 26: Zero-order double cascade

```
directive sample 0.03
directive plot !a; !b

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

let Amp_hi(a:chan, b:chan) =
  do !b; delay@1.0; Amp_hi(a,b)
  or delay@1.0; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; ?a; Amp_hi(a,b)

run 1000 of Amp_lo(a,b)

let A() = !a; delay@1.0; A()
run 2000 of A()
```

Figure 27: Simple inverter

```
directive sample 110.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Inv_hi(a:chan, b:chan) =
  do !b; Inv_hi(a,b)
  or ?a; Inv_lo(a,b)
and Inv_lo(a:chan, b:chan) =
  delay@1.0; Inv_hi(a,b)

run 100 of (Inv_hi(a,b) | Inv_lo(b,c))

let clock(t:float, tick:chan) =
  (* sends a tick every t time *)
  (val ti = t/100.0 val d = 1.0/ti
   (* by 100-step erlang timers *)
   let step(n:int) = if n<=0
     then !tick; clock(t,tick)
     else delay@d; step(n-1)
   run step(100))
let S1(a:chan, tock:chan) =
  do !a; S1(a,tock) or ?tock; ()
let SN(n:int, t:float, a:chan, tick:chan, tock:chan) =
  if n=0 then clock(t, tock)
  else ?tick; (S1(a,tock) | SN(n-1,t,a,tick,tock))
let raisingfalling(a:chan, n:int, t:float) =
  (new tick:chan new tock:chan
   run (clock(t,tick) | SN(n,t,a,tick,tock)))

run raisingfalling(a,100,0.5)
```

```
directive sample 15.0 1000
directive plot !a; !b; !c; !d; !e; !f

new a@1.0:chan new b@1.0:chan new c@1.0:chan
new d@1.0:chan new e@1.0:chan new f@1.0:chan

let Inv_hi(a:chan, b:chan) =
  do !b; Inv_hi(a,b)
  or ?a; Inv_lo(a,b)
and Inv_lo(a:chan, b:chan) =
  delay@1.0; Inv_hi(a,b)

run 100 of (Inv_lo(a,b) | Inv_lo(b,c)
| Inv_lo(c,d) | Inv_lo(d,e) | Inv_lo(e,f))
```

```
directive sample 2.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Inv_hi(a:chan, b:chan) =
  do !b; Inv_hi(a,b)
  or ?a; Inv_lo(a,b)
and Inv_lo(a:chan, b:chan) =
  delay@1.0; Inv_hi(a,b)

run 100 of (Inv_hi(a,b) | Inv_lo(b,c) | Inv_lo(c,a))
```

Figure 28: Feedback inverter

```
directive sample 110.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Inv_hi(a:chan, b:chan) =
  do !b; Inv_hi(a,b) or ?a; Inv_lo(a,b)
and Inv_lo(a:chan, b:chan) =
  do delay@1.0; Inv_hi(a,b)
  or ?b; Inv_hi(a,b)

run 100 of (Inv_hi(a,b) | Inv_lo(b,c))

let raisingfalling(a:chan, n:int, t:float) =
  ... see code for Figure 27

run raisingfalling(a,100,0.5)
```

```
directive sample 1.0 1000
directive plot !a; !b; !c; !d; !e; !f

new a@1.0:chan new b@1.0:chan new c@1.0:chan
new d@1.0:chan new e@1.0:chan new f@1.0:chan

let Inv_hi(a:chan, b:chan) =
  do !b; Inv_hi(a,b) or ?a; Inv_lo(a,b)
and Inv_lo(a:chan, b:chan) =
  do delay@1.0; Inv_hi(a,b)
  or ?b; Inv_hi(a,b)

run 100 of (Inv_lo(a,b) | Inv_lo(b,c)
| Inv_lo(c,d) | Inv_lo(d,e) | Inv_lo(e,f))
```

```
directive sample 2.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Inv_hi(a:chan, b:chan) =
  do !b; Inv_hi(a,b) or ?a; Inv_lo(a,b)
and Inv_lo(a:chan, b:chan) =
  do delay@1.0; Inv_hi(a,b)
  or ?b; Inv_hi(a,b)

run 100 of (Inv_hi(a,b) | Inv_lo(b,c) | Inv_lo(c,a))
```

Figure 29: Double-height inverter

```
directive sample 110.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
```

```
let Inv2_hi(a:chan, b:chan) =
  do !b; Inv2_hi(a,b) or ?a; Inv2_mi(a,b)
and Inv2_mi(a:chan, b:chan) =
  do delay@1.0; Inv2_hi(a,b)
  or ?a; Inv2_lo(a,b)
and Inv2_lo(a:chan, b:chan) =
  delay@1.0; Inv2_mi(a,b)

run 100 of (Inv2_hi(a,b) | Inv2_lo(b,c))

let raisingfalling(a:chan, n:int, t:float) =
  ... see code for Figure 27

run raisingfalling(a,100,0.5)
```

```
directive sample 15.0 1000
directive plot !a; !b; !c; !d; !e; !f

new a@1.0:chan new b@1.0:chan new c@1.0:chan
new d@1.0:chan new e@1.0:chan new f@1.0:chan

let Inv2_hi(a:chan, b:chan) =
  do !b; Inv2_hi(a,b) or ?a; Inv2_mi(a,b)
and Inv2_mi(a:chan, b:chan) =
  do delay@1.0; Inv2_hi(a,b)
  or ?a; Inv2_lo(a,b)
and Inv2_lo(a:chan, b:chan) =
  delay@1.0; Inv2_mi(a,b)

run 100 of (Inv2_lo(a,b) | Inv2_lo(b,c)
| Inv2_lo(c,d) | Inv2_lo(d,e) | Inv2_lo(e,f))
```

```
directive sample 2.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Inv2_hi(a:chan, b:chan) =
  do !b; Inv2_hi(a,b) or ?a; Inv2_mi(a,b)
and Inv2_mi(a:chan, b:chan) =
  do delay@1.0; Inv2_hi(a,b)
  or ?a; Inv2_lo(a,b)
and Inv2_lo(a:chan, b:chan) =
  delay@1.0; Inv2_mi(a,b)

run 100 of (Inv2_hi(a,b) | Inv2_lo(b,c) | Inv2_lo(c,a))
```

Figure 30: Double-height feedback inverter

```
directive sample 110.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Inv2_hi(a:chan, b:chan) =
  do !b; Inv2_hi(a,b) or ?a; Inv2_mi(a,b)
and Inv2_mi(a:chan, b:chan) =
  do delay@1.0; Inv2_hi(a,b)
  or ?a; Inv2_lo(a,b)
  or ?b; Inv2_hi(a,b)
and Inv2_lo(a:chan, b:chan) =
  do delay@1.0; Inv2_mi(a,b)
  or ?b; Inv2_mi(a,b)

run 100 of (Inv2_hi(a,b) | Inv2_lo(b,c))

let raisingfalling(a:chan, n:int, t:float) =
  ... see code for Figure 27

run raisingfalling(a,100,0.5)
```

```
directive sample 1.0 1000
directive plot !a; !b; !c; !d; !e; !f

new a@1.0:chan new b@1.0:chan new c@1.0:chan
new d@1.0:chan new e@1.0:chan new f@1.0:chan

let Inv2_hi(a:chan, b:chan) =
  do !b; Inv2_hi(a,b) or ?a; Inv2_mi(a,b)
and Inv2_mi(a:chan, b:chan) =
  do delay@1.0; Inv2_hi(a,b)
  or ?a; Inv2_lo(a,b)
  or ?b; Inv2_hi(a,b)
and Inv2_lo(a:chan, b:chan) =
  do delay@1.0; Inv2_mi(a,b)
  or ?b; Inv2_mi(a,b)

run 100 of (Inv2_lo(a,b) | Inv2_lo(b,c)
| Inv2_lo(c,d) | Inv2_lo(d,e) | Inv2_lo(e,f))
```

```
directive sample 2.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Inv2_hi(a:chan, b:chan) =
  do !b; Inv2_hi(a,b) or ?a; Inv2_mi(a,b)
and Inv2_mi(a:chan, b:chan) =
  do delay@1.0; Inv2_hi(a,b)
  or ?a; Inv2_lo(a,b)
  or ?b; Inv2_hi(a,b)
and Inv2_lo(a:chan, b:chan) =
  do delay@1.0; Inv2_mi(a,b)
  or ?b; Inv2_mi(a,b)

run 100 of (Inv2_hi(a,b) | Inv2_lo(b,c) | Inv2_lo(c,a))
```

Figure 31: Or and And

```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Or_hi(a:chan, b:chan, c:chan) =
  do !c; Or_hi(a,b,c) or delay@del; Or_lo(a,b,c)
and Or_lo(a:chan, b:chan, c:chan) =
  do ?a; Or_hi(a,b,c) or ?b; Or_hi(a,b,c)

run 1000 of Or_lo(a,b,c)

let clock(t:float, tick:chan) =
  (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) = if n<=0
     then !tick; clock(t, tick)
     else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) =
  do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

let many(n:float, p:proc()) =
if n<=0.0 then () else (p() | many(n-1.0, p))

let BoolInputs(n:float, nt:float, m:float, mt:float) =
  (let Sig_a() =
     (new tick:chan run (clock(nt,tick) | S_a(tick)))
   let Sig_b() =
     (new tick:chan run (clock(mt,tick) | S_b(tick)))
   run many(n, Sig_a)
   run many(m, Sig_b))

run BoolInputs(100.0, 4.0, 100.0, 2.0)
```

```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let And_hi(a:chan, b:chan, c:chan) =
  do !c; And_hi(a,b,c) or delay@del; And_lo_a(a,b,c)
and And_lo_a(a:chan, b:chan, c:chan) =
  do ?a; And_hi(a,b,c) or delay@del; And_lo_b(a,b,c)
and And_lo_b(a:chan, b:chan, c:chan) =
  ?b; And_lo_a(a,b,c)

run 1000 of And_lo_b(a,b,c)

let BoolInputs(n:float, nt:float, m:float, mt:float) =
  ... see code for Figure 31

run BoolInputs(100.0, 4.0, 100.0, 2.0)
```

Figure 32: Imply and Xor

```
directive sample 15.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0
```

```
let Imply_hi_a(a:chan, b:chan, c:chan) =
  do !c; Imply_hi_a(a,b,c) or ?a; Imply_lo(a,b,c)
and Imply_hi_b(a:chan, b:chan, c:chan) =
  do !c; Imply_hi_b(a,b,c)
  or delay@del; Imply_lo(a,b,c)
and Imply_lo(a:chan, b:chan, c:chan) =
  do ?b; Imply_hi_b(a,b,c)
  or delay@del; Imply_hi_a(a,b,c)

run 1000 of Imply_lo(a,b,c)

let BoolInputs(n:float, nt:float, m:float, mt:float) =
  ... see code for Figure 31

run BoolInputs(100.0, 4.0, 100.0, 2.0)
```

```
directive sample 20.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Xor_hi_a(a:chan, b:chan, c:chan) =
  do !c; Xor_hi_a(a,b,c) or ?b; Xor_lo_ab(a,b,c)
  or delay@1.0; Xor_lo_a(a,b,c)
and Xor_hi_b(a:chan, b:chan, c:chan) =
  do !c; Xor_hi_b(a,b,c) or ?a; Xor_lo_ab(a,b,c)
  or delay@1.0; Xor_lo_b(a,b,c)
and Xor_lo_a(a:chan, b:chan, c:chan) =
  do ?a; Xor_hi_a(a,b,c) or ?b; Xor_lo_ab(a,b,c)
and Xor_lo_b(a:chan, b:chan, c:chan) =
  do ?b; Xor_hi_b(a,b,c) or ?a; Xor_lo_ab(a,b,c)
and Xor_lo_ab(a:chan, b:chan, c:chan) =
  do delay@1.0; Xor_hi_a(a,b,c)
  or delay@1.0; Xor_hi_b(a,b,c)

run 500 of (Xor_lo_a(a,b,c) | Xor_lo_b(a,b,c))


let BoolInputs(n:float, nt:float, m:float, mt:float) =
  ... see code for Figure 31

run BoolInputs(100.0, 8.0, 100.0, 4.0)
```

Figure 34: Complexation/decomplexation

```
directive sample 0.005
directive plot Af(); Ab(); Bf(); Bb()

val mu = 1.0    val lam = 1.0
new a@mu:chan(chan)

let Af() = (new n@lam:chan run !a(n); Ab(n))
and Ab(n:chan) = !n; Af()

let Bf() = ?a(n); Bb(n)
and Bb(n:chan) = ?n; Bf()

run (1000 of Af() | 500 of Bf())
```

Figure 35: Enzymatic reactions

```
directive sample 0.05 1000
directive plot Ef(); Eb(); Sf(); Sb(); P()

val k1 = 1.0    val km1 = 1.0    val k2 = 100.0
new a@k1:chan(chan,chan)

let P() = ()

let Ef() =
   (new n@km1:chan new m@k2:chan
    run !a(n,m); Eb(n,m))
and Eb(n:chan,m:chan) =
  do !n; Ef() or !m; Ef()

let Sf() = ?a(n,m); Sb(n,m)
and Sb(n:chan,m:chan) =
  do ?n; Sf() or ?m; P()

run (1000 of Ef() | 2000 of Sf())
```

Figure 36: Homodimerization

```
directive sample 0.005 10000
directive plot Af(); Ai()

new a@1.0:chan(chan)

let Af() =
  (new n@1.0:chan
   run do ?a(m); Ai(m) or !a(n); Ao(n))

and Ai(n:chan) = ?n; Af()

and Ao(n:chan) = !n; Af()

run 1000 of Af()
```

Figure 37: Bidirectional polymerization

```
directive sample 1000.0
directive plot ?count
(* directive plot Af(); Al(); Ar(); Ab() *)

type Link = chan(chan)
type Barb = chan

val lam = 1000.0 (* set high for better counting *)
val mu = 1.0
new c@mu:chan(Link)
new enter@lam:chan(Barb)
new count@lam:Barb

let Af() =
  (new rht@lam:Link run
   do !c(rht); Ar(rht)
   or ?c(lft); Al(lft))

and Al(lft:Link) =
  (new rht@lam:Link run
   !c(rht); Ab(lft,rht))

and Ar(rht:Link) =
   ?c(lft); Ab(lft,rht)

and Ab(lft:Link, rht:Link) =
  do ?enter(barb); (?barb | !rht(barb))
  or ?lft(barb); (?barb | !rht(barb))
(* each Abound waits for a barb, exhibits it, and
passes it to the right so we can plot number of Abound
in a ring *)

let clock(t:float, tick:chan) =
  (* sends a tick every t time *)
  (val ti = t/1000.0 val d = 1.0/ti
   let step(n:int) = if n<=0
     then !tick; clock(t,tick)
     else delay@d; step(n-1)
   run step(1000))

new tick:chan
let Scan() = ?tick; !enter(count); Scan()

run 1000 of Af()
run (clock(100.0, tick) | Scan())
```

Figure 39: Actin-like polymerization

```
directive sample 1000.0
directive plot Af(); Al(); Ar(); Ab()

val lam = 1.0 (* dissoc *)
val mu = 1.0 (* assoc *)
new c@mu:chan(chan)

let Af() =
  (new lft@lam:chan run
   do !c(lft); Al(lft) or ?c(rht); Ar(rht))

and Al(lft:chan) =
   do !lft; Af() or ?c(rht); Ab(lft,rht)

and Ar(rht:chan) = ?rht; Af()

and Ab(lft:chan, rht:chan) = !lft; Ar(rht)

run 1000 of Af()
```