Artificial Biochemistry

Luca Cardelli

Microsoft Research

Abstract

We model chemical and biochemical systems as collectives of interacting stochastic automata, with each automaton representing a molecule that undergoes state transitions. In this *artificial biochemistry*, automata interact by the equivalent of the law of mass action. We investigate several simple but intriguing automata collectives by stochastic simulation and by ODE analysis.

1 Introduction

Macromolecules

Molecular biology investigates the structure and function of biochemical systems starting from their basic building blocks: *macromolecules*. A macromolecule is a large, complex molecule (a protein or a nucleic acid) that usually has inner mutable state and external activity. Informal explanations of biochemical events trace individual macromolecules through their state changes and their interaction histories: a macromolecule is endowed with an *identity* that is retained through its transformations, even through changes in molecular energy and mass. A macromolecule, therefore, is qualitatively different from the *small molecules* of inorganic chemistry. Such molecules are *stateless*: in the standard notation for chemical reactions they are seemingly created and destroyed, and their atomic structure is used mainly for the bookkeeping required by the conservation of mass.

Attributing identity and state transitions to molecules provides more than just a different way of looking at a chemical event: it solves a fundamental difficulty with chemical-style descriptions. Each macromolecule can have a huge number of internal states, exponentially with respect to its size, and can join with other macromolecules to from even larger state configurations, corresponding to the product of their states. If each molecular state is to be represented as a stateless chemical species, transformed by chemical reactions, then we have a huge explosion in the number of species and reactions with respect to the number of different macromolecules that actually, physically, exist. Moreover, macromolecules can join to each other indefinitely, resulting in situations corresponding to infinite sets of chemical reactions among infinite sets of different chemical species. In contrast, the description of a biochemical system at the level of macromolecular states and transitions remains finite: the unbounded complexity of the system is implicit in the *potential* molecular interactions, but does not have to be written down explicitly. Molecular biology textbooks widely adopt this finite description style, at least for the purpose of illustration.

Many proposal now exist that aim to formalize the combinatorial complexity of biological systems without a corresponding explosion in the notation. One of the earliest can be found in [3] (which inspired the title of this article), where an artificial formal frameworks is used to get insights into natural systems. More recently, the descriptive paradigm in systems biology has become that of *programs as models* [7][11][19]. Macromolecules, in particular, are seen as *stateful concurrent agents* that *interact with each other* through a *dynamic interface*. While this style of descriptions is (like many others) not quite accurate

at the atomic level, it forms the basis of a formalized and growing body of biological knowledge.

The complex chemical structure of a macromolecule is thus commonly abstracted into just internal states and potential interactions with the environment. Each macromolecule forms, symmetrically, part of the environment for the other macromolecules, and can be described without having to describe the whole environment. Such an *open system* descriptive style allows modelers to extend systems by composition, and is fundamental to avoid enumerating the whole combinatorial state of the system (as one ends up doing in *closed systems* of chemical reactions). The programs-as-models approach is growing in popularity with the growing modeling ambitions in systems biology, and is, incidentally, the same approach taken in the organization of software systems. The basic problem and the basic solution are similar: programs are finite and compact models of potentially unbounded state spaces.

Molecules as Automata

At the core, we can therefore regard a macromolecule as some kind of automaton, characterized by a set of internal states and a set of discrete transitions between states driven by external interactions. We can thus try to handle molecular automata by some branch of automata theory and its outgrowths: cellular automata, Petri nets, and process algebra. The peculiarities of biochemistry, however, are such that until recently one could not easily pick a suitable piece of automata theory off the shelf.

Many sophisticated approaches have now been developed, and we are particularly fond of stochastic process algebra [18]. In this paper, however, we do our outmost to remain within the bounds of a much simpler theory. We go back, in a sense, to a time before cellular automata, Petri nets and process algebra, which all arose from the basic intuition that automata should interact with each other. Our main criterion is that, as in finite-state automata, we should be able to easily and separately *draw* the individual automata, both as a visual aid to design and analysis, and to emulate the illustration-based approach found in molecular biology textbooks. As a measure of success, in this paper we draw a large number of examples.

Technically, we place ourselves within a small fragment of a well-know process algebra (stochastic π-calculus), but the novelty of the application domain, namely the "mass action" behavior of large numbers of "well-mixed" automata, demands a broader outlook. We rely on the work in [1][2] for foundations and in-depth analysis. In this paper we aim instead 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 obtained by large populations of simple automata. Our automata drawings are precise, but the only formalization can be found in the corresponding process algebra scripts in the Appendix.

Stochastic Automata Collectives

With those aims, we investigate *stochastic automata collectives*. 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 automata interactions. It is also not quite the situation with cellular automata, because our automata are interacting, but not necessarily on a regular grid. And it is not quite the situation in process algebra, because we are interested in the behavior of collectives, not of individuals. And in contrast to Petri nets, we model separate parts of a system separately. Similar frameworks have been investigated under the headings of collectives [23], sometimes including stochasticity [12]. The broad area of computer network analysis is also relevant; see [8] for a bridge between that and stochastic automata.

By *stochastic* we mean that automata interactions have rates. These rates induce a quantitative semantics for the behavior of collectives, and allow them to mimic chemical kinetics. Chemical systems are, physically, formed by the *stochastic* interactions of *discrete* particles. For large number of particles it is usually possible to consider them as formed by *continuous* quantities that evolve according to *deterministic* laws, and to analyze them by ordinary differential equations (ODEs). However, one should keep in mind that continuity is an abstraction, and that sometimes it is not even a correct limit approximation. In biochemistry, the stochastic discrete approach is particularly appropriate because cells often contain very low numbers of molecules of critical species: that is a situation where continuous models may be misleading. Stochastic automata collectives are hence directly inspired by biochemical systems, which are sets of interacting macromolecules, whose stochastic behavior ultimately derives from molecular dynamics. Some examples of the mismatch between discrete and continuous models are discussed at the end of Section 3.

Paper Outline

In Section 2 we introduce the notion of stochastic interacting automata. In Section 3 we explore a number of examples inspired by chemical kinetics, leading to the implementation of basic analog and digital devices. In Section 4 we describe more sophisticated automata, which are more suitable for biochemical modeling. In Section 5 we discuss further related work, and conclude. The Appendix contains the simulation scripts for the figures. The figures are in color (the layout and the narrative offer color-neutral hints, but we encourage reading the digital version of this paper). Auxiliary materials (including magnifiable figures and editable simulation scripts) are at <http://LucaCardelli.name>.

2 Interacting Automata

2.1 Automata Reactions

We begin by focusing 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, each with three *possible states* (circles) and each with a *current state* (yellow): initially A_1, B_1, C_1, respectively. *Transitions* change the current state of an automaton to one of its possible states: they are drawn as thick gray arrows (solid or dashed). *Interactions* between separate automata are drawn as thin red dashed arrows. *Collectives* consist of populations of automata, e.g. 100×A, 200×B and 300×C.

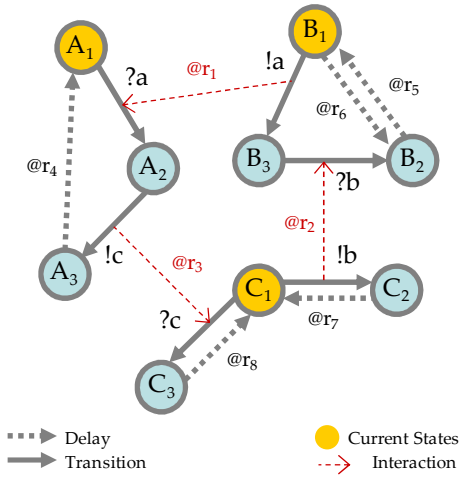

Figure 1 Interacting automata

There are two possible kinds of *reactions* that can cause an automaton to take a transition and change its current state; each reaction changes the situation depicted on the left of Figure 2 to the situation on the right, within some larger context. First, from its current state, an automaton can spontaneously execute a *delay* transition (dashed gray arrow) resulting in a change of state of that automaton. Second, an automaton can jointly execute an *interaction* (thin red dashed arrow) with a separate automaton. In an interaction, one automaton executes an *input* (?), and the other an *output* (!) on a common *channel* (a). A channel is an abstraction (just a name) for any interaction surface or mechanism, and input/output are abstraction for any kind of interaction complementarity. An actual interaction can happen only if both automata are in a current state such that the interaction is enabled along complementary transitions (solid gray arrows); if the interaction happens, then both automata change state simultaneously. The system of automata in Figure 1, for example, could go through the following state changes: $A_1,B_1,C_1 \rightarrow_{(a)} A_2,B_3,C_1 \rightarrow_{(b)} A_2,B_2,C_2 \rightarrow A_2,B_2,C_1 \rightarrow_{(c)} A_3,B_2,C_3 \rightarrow A_1,B_2,C_3 \rightarrow A_1,B_2,C_1 \rightarrow A_1,B_1,C_1$.

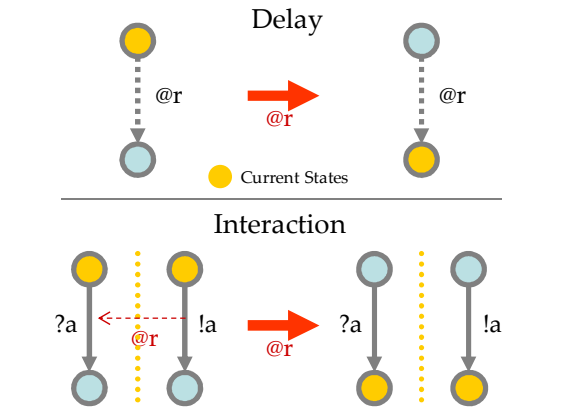

Figure 2 Automata reactions

Each reaction fires at (@) a rate r. In the case of interaction, the rate is associated to a channel (i.e., interactions have rates, but input/output transitions do *not* have rates of their own). Reaction rates determine, stochastically, the choice of the next reaction to execute, and also determine the time spent between reactions [4][15]. In particular, the probability of an enabled reaction with rate r occurring within time t is given by an exponential distribution $F(t) = 1\text{-}e^{-rt}$ with mean 1/r. A Continuous Time Markov Chain (CTMC) can be extracted from an automata collective [1]: a state of such a CTMC is a multiset of the automata current states for the population, and a transition in the CTMC has a rate that is the sum of the rates of all the reactions connecting two states.

2.2 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 usually set our reaction rates to 1.0 (and in that case omit them in figures) not because rates are unimportant, but because rich behavior can be obtained already by changing the automata structure: rate variation is a further dimension of complexity. The 1.0 rates still determine the pacing of the system in time.

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 symmetric reactions). If we have two automata in state B, one will similarly move to state A. If we have one in state A and one in state B, then no interactions are possible and the system is stable.

We call such automata *celebrities* because they aim to be different: if one of them "sees" another celebrity in the same state, it changes state. How will a population of celebrities behave? Starting with 100 celebrities in state A and 100 in state B, the stochastic simulation in Figure 3 (obtained by the techniques in [15] and Appendix) shows that a 50/50 noisy equilibrium is maintained. Moreover, the system is live: individual celebrities keep changing state.

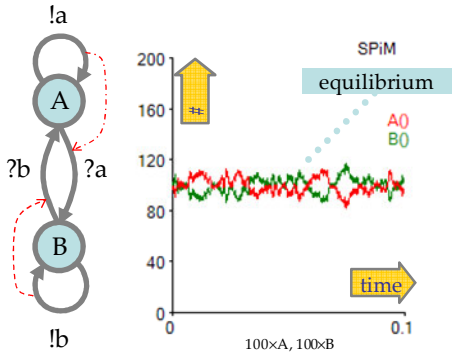

Figure 3 Celebrity automata

The possible interactions between celebrities are indicated in Figure 3 by thin-red-dashed *interaction arrows* between transitions on the same automaton. Remember, however, that an automaton can never interact with itself: this abuse of notation refers unambiguously to interactions between distinct automata in a collective. The possible interactions should more properly be read out from the complementary transition labels. The transition labels emphasize the *open system* interactions with all possible environments, while the interaction arrows emphasize the *closed system* interactions in a given collective.

Figure 4 shows more explicitly the possible interactions in a population of 5 celebrity automata, of which 3 are in state A and 2 are in state B. Note that since there are more "a interactions" than "b interactions", and their rates are equal, at this point in time an "a interaction" between a pair of automata is more likely. These considerations about the likelihood of interactions are part of the stochastic simulation algorithm, and also lead to the chemical *law of mass action* for large populations [24].

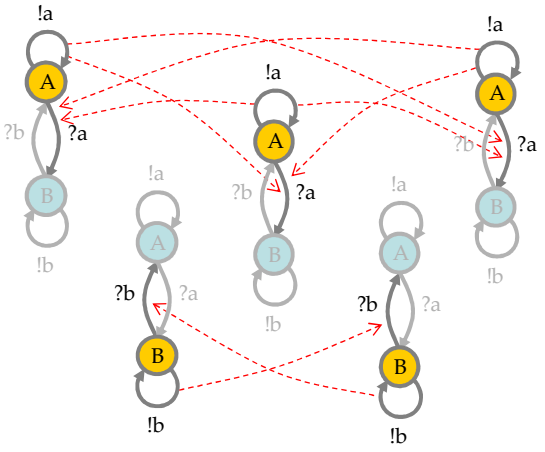

Figure 4 Possible interactions

Let us now consider a different two-state automaton shown in Figure 5. 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 also interact on channel b and both move to state B.

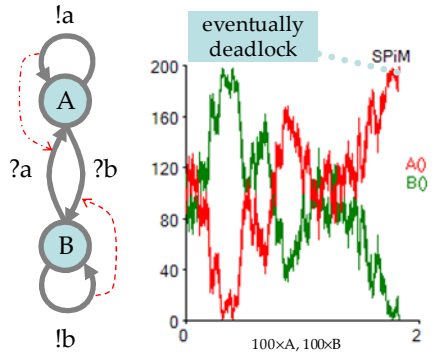

Figure 5 Groupie automata

We call such automata *groupies* because they aim to be similar: two groupies in different states will switch to equal states. How will a population of groupies behave? In Figure 5 we start with 100 A and 100 B: the system evolves through a bounded random walk, and the outcome remains uncertain

till the very end. Eventually, though, the groupies form a single homogeneous population of all A or all B, and the system is then dead: no automaton can change state any further. Different runs of the simulation may randomly produce all A or all B: the system is *bistable*.

2.3 Mixed Populations

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 6, 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.

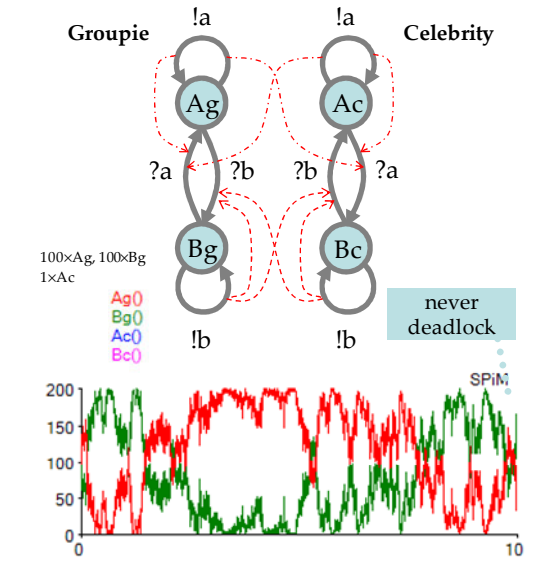

Figure 6 Both together

An important lesson here is that an arbitrarily small, but non-zero, number of celebrities can transform the *macroscopic* groupie behavior from a system that always eventually deadlocks, to a system that never deadlocks. We can also replace celebrities with simpler *doping* automata (Figure 7) that have the same effect of destabilizing groupie collectives.

We now change the structure of the groupie automaton by introducing intermediate states on the transitions between A and B, while still keeping all reactions rates at 1.0. In Figure 7, each groupie in state A must find two groupies (left) or three groupies (right) 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. Both systems include doping automata (center) to avoid population deadlocks.

The additional intermediate states produce a striking change in behavior with respect to Figure 6: from complete randomness to irregular (left) and then regular (right) oscillations. The peaks in the plots of Figure 7 are stochastic both in height and in width, and occasionally one may observe some miss-steps, but they clearly alternate. The transformation in behavior, obtained by changes in the structure of individual automata, is certainly remarkable (and largely independently on rate values). Moreover, the oscillations depend critically on the tiny perturbations introduced by doping: without them, the system typically stops on its first cycle.

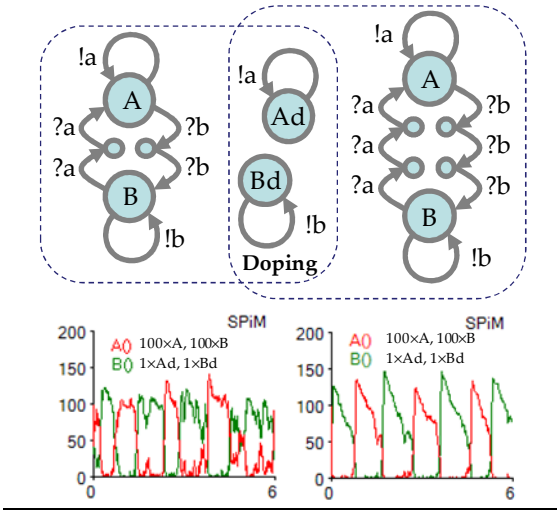

Figure 7 Hysteric groupies

The morale from these examples is that the collective behavior of even the simplest interactive automata can be rich and surprising. Macroscopic behavior "emerges" apparently unpredictably from the structure of the components, and even a tiny number of components can have macroscopic effects. The question then arises, how can we relate the macroscopic behavior to the microscopic structure? In the following section we continue looking at examples, many of which can be analyzed by continuous methods. We return to discussing the behavior of groupies at the end of Section 3.

3 The Chemistry of Automata

3.1 Concentration

The first few chapters of a chemical kinetics textbook [9] usually include a discussion of the *order* of a chemical reaction. In particular, a reaction of the

form $A \rightarrow^{r} C$ is first-order, and a reaction of the form $A+B \rightarrow^{r} C$ is second-order. The terminology derives from the order of the polynomials of the associated differential equations. In the first case, the *concentration* of A, written [A], follows the exponential decay law $d[A]/dt = -r[A]$, where the right-hand side is a first-order term with coefficient –r (r being the base reaction rate). In the second case, the concentration of A follows the mass action law $d[A]/dt = -r[A][B]$, where the right-hand side is a second-order term. (In the sequel, we use $[A]^{\bullet}$ for $d[A]/dt$.)

Our automata collectives should match these laws of chemical kinetics, at least in large number approximations. But what should be the meaning of "a concentration of discrete automata states" that would follow such laws? That sounds puzzling, but is really the same question as the meaning of "a concentration of discrete molecules" in a chemical system. One has to first fix a volume of interaction (assumed filled with, e.g., water), and then divide the discrete number of molecules by that volume, obtaining a concentration that is regarded (improperly) as a continuous quantity. Along these lines, a relationship between automata collectives and differential equations is studied formally in [1].

Under biological conditions, common simplifying assumptions are that the volume of interaction is constant, that the volume is filled with a liquid solution with constant temperature and pressure, and that the solution is *dilute* and *well-mixed*. The dilution requirement is a limit on maximum chemical concentrations: there should be enough water that the collisions of chemicals with water are more frequent than among themselves. The well-mixed requirement means that diffusion effects are not important. Together, these physical assumptions justify the basic mathematical assumption: the probability of any two non-water molecules colliding (and reacting) in the near future is independent of their current position, so that we need to track only the number (or concentration) of molecules, and not their positions. Three-way collisions are too unlikely to matter [4].

Therefore, we assume that our automata interact within a fixed volume V, and we use a scaling factor $\gamma = N_A V$, where N_A is, in chemistry, Avogadro's number. For most purposes we can set $\gamma = 1.0$, which means that we are considering a volume of size $1/N_A$ liters. However, keeping γ symbolic helps in performing any scaling of volume (and hence of number of automata) that may be needed.

With these basic assumptions, we next analyze the reaction orders that are available to our collectives, and the effect of reaction order on kinetics.

3.2 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 $[A]_t = \#A_t/\gamma$ is the concentration of automata in state A at time t, and $\#A_0$ is the initial number of automata in state A. The concentration [A] is a continuous quantity, and is more properly related to the *expectation* of the discrete number #A having a certain value [24]. The rate of change for the reaction (assuming $A' \neq A$) is then the derivative of [A], written $[A]^{\bullet} = -r[A]$. The profile of the reaction is an exponential decay at rate r: $[A]_0 e^{-rt}$ (see S_1 in Figure 9).

Figure 8 First order reactions

A sequence of exponential decays produces an Erlang distribution, as seen in Figure 9 (many biological processes, like the sequential transcription and translation of DNA, behave similarly). Initially, we have C=10000 automata in state S_1. The occupation of the initial state S_1 is an exponential decay, the occupation of the intermediate states S_i is the Erlang distribution of shape parameter i, and the occupation of the final state is the cumulative Erlang distribution of shape parameter 10.

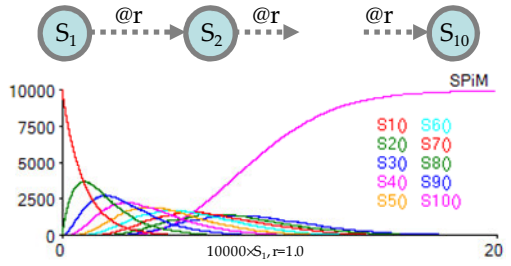

Figure 9 Sequence of delays

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 9 and we scale down the vertical axis by a factor of 10, we obtain the same plot, to the original time 20. Meaning that the "speed of the systems" is the same as before, and since there are 10 times more reactions in the same time span, the "execution rate" is 10 times higher.

3.3 Second Order Reactions

As discussed in Section 2, 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 state A performs an input ?a and moves to state A′; the other automaton in state B performs an output !a and moves to state B′. This interaction can be written as a chemical reaction $A+B \to^{r\gamma} A'+B'$ (Figure 10, top), where r is the fixed rate assigned to the interaction channel, and rγ is the volume scaling for A+B reactions [24] (to scale the volume γ to n·γ, we must scale $\#X_0$ to $n\cdot\#X_0$ to keep $[X]_0$ the same, and r to r/n to keep rates (r/n)n·γ the same). The rate law, given by the law of mass action, is $-r\gamma[A][B]$, because each automaton in the population of current states [A] can interact with each automaton in the population of current states [B], and the derivatives (assuming A,B,A′,B′ are distinct) are $[A]^\bullet = [B]^\bullet = -r\gamma[A][B]$.

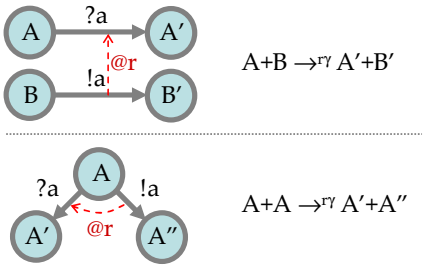

Figure 10 Second order reactions

A different situation arises, though, if the interaction happens within a homogeneous 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″ (Figure 10, bottom). Then, every automaton in state A can interact with every other automaton in state A in *two* symmetric ways; hence the rate r must be doubled to 2r. The volume scaling for A+A reactions is (2r)γ/2 = rγ [24]. The chemical reaction is then $A+A \to^{r\gamma} A'+A''$, whose rate law is $-r\gamma[A]^2$. The rate of change of [A] (assuming $A' \neq A \neq A''$) is $[A]^\bullet = -2r\gamma[A]^2$, since two A are lost each time.

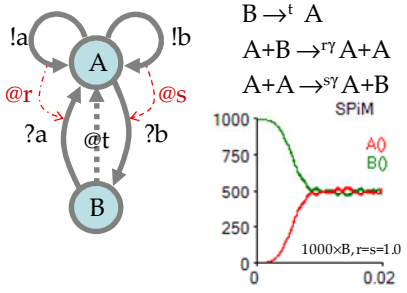

Figure 11 All 3 reactions in 1 automaton

In Figure 11 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. This shows that the dynamics of all orders of reactions can become intermingled in a single automaton.

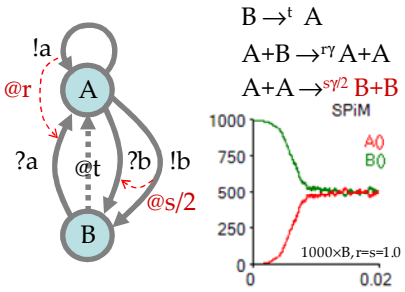

Figure 12 Same behavior

In order to compare the behavior of different automata collectives, we must in general go beyond the corresponding chemical reactions, and we must instead compute the corresponding ODEs (which can be obtained from the chemical reactions). For example, the automaton in Figure 12 has a different pattern of interactions and rates, different chemical reactions, but the same ODEs as the one in Figure 11. In both cases, $[A]^\bullet = -[B]^\bullet = t[B] + r\gamma[A][B] - s\gamma[A]^2$, but note that the b rate in Figure 12 is set to s/2 in order to obtain the same rate of decrease in A population and rate of increase in B population as in Figure 11, given the differences in the corresponding chemical reactions.

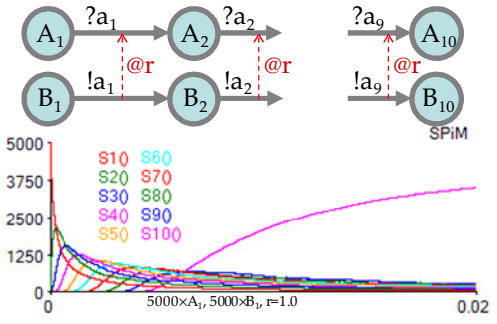

Figure 13 Sequence of interactions

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 13, 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. 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 13 is about 500 times faster than the one in Figure 9 in

reaching 75% of input level in its final state, even though it has the same rates and the same number of automata. Second order reactions "go faster".

3.4 Zero Order Reactions

As we have just seen, the two basic kinds of automata interactions, which correspond to the two basic kinds of chemical reactions, lead to collective dynamics characterized by various kinds of curves. But what if we wanted to build a population that spontaneously (from fixed initial conditions) increases or decreases at a constant rate, as in Figure 14? We can consider this as a programming exercise in automata collectives: *can we make straight lines?* This question will lead us to implementing a basic analog component: a signal comparator.

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 (a constant derivative), meaning that the "execution rate" is constant, and hence the "speed of the system" gets slower when more ingredients are added. Zero order reactions are not built into chemistry (except as spontaneous creation reactions $0 \rightarrow^{r} A$), but can be approximated by chemical means. The main biochemical methods of obtaining zero order reactions is a special case of enzyme kinetics, when enzymes are saturated.

Real enzyme kinetics corresponds to a more sophisticated notion of automata: both are treated in Section 4. For now, 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 simplified way.

Consider the system of Figure 14. Here E is the (*pseudo-*) enzyme, S is the substrate being transformed with the help of E, and P is the product resulting from the transformation. The state ES represents an enzyme that is "temporarily unavailable" because it has just transformed some S into some P, and needs time to recover (ES does not represent an E molecule bound to an S molecule: it is just a different state of the E molecule alone).

This system exhibits zero order kinetics (i.e., a constant slope), 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, independently of the instantaneous quantity of S. That happens because E becomes maximally busy, and effectively processes S sequentially. Adding more enzyme (up to a point) will just increase the ES population: 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].

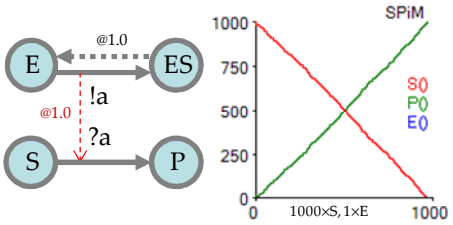

Figure 14 Zero order reactions

To explain the reason for the constant slope, and to make the connection to enzyme kinetics precise, we now mimic the standard derivation of Michaelis-Menten kinetics [9]. The reactions for the system in Figure 14 are:

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

The corresponding ODEs are:

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

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

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

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:

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

which describes $[P]^{\bullet}$ just in terms of [S] and two constants. 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 14 at $\gamma = 1$ we have $K_m = 1$, $[S]_0 = 1000$, $V_{max} = 1$, and hence $[P]^{\bullet} \approx 1$, as shown in the simulation.

The chemical reactions for Figure 14 are significantly different from the standard enzymatic reactions, where P is produced after the breakup of ES,

and not before as here. 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 of ES back to E+S is negligible, that is, for good enzymes.

3.5 Ultrasensitivity

Zero-order kinetics can be used, rather paradoxically, to obtain sudden non-linearity or switching behavior. Let us first compare the behavior of directly competing enzymes in zero-order and second-order kinetics. For conciseness we now depict a pair of states E,ES (as in Figure 14) as a single state E with a solid/dashed arrow representing the transition through the now hidden state ES (Figure 15).

At the top of Figure 15, in zero-order regime, a fixed quantity of F is competing against a linearly growing quantity of E. The circuit is essentially computing the subtraction [F]-[E] (the rest being sequestered in the hidden "unavailable" states): the E quantity is neutralized until it can neutralize and then exceed the F quantity. At the bottom we have almost the same system, except in second-order regime: the result of the competition is quite different because neither quantity can be sequestered.

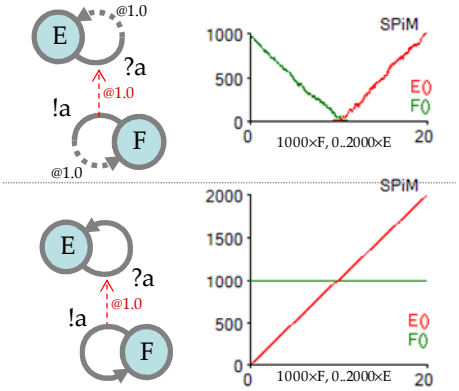

Figure 15 Subtraction (top)

On that basis, we now reproduce the peculiar phenomenon of ultrasensitivity in zero-order regime [13], confirming that our simplified kinetics, while not agreeing with enzyme kinetics at the microscopic level, still manages to reproduce some of its macroscopic effects: the core of the matter is the zero-order regime of operation. In an ultrasensitivity situation, a minor switch in relative enzyme quantities creates a much amplified and sudden switch in two other quantities. In Figure 16, we start with a fixed amount (=100) of enzyme F, which is holding the S vs. P equilibrium in the S state (at S=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 (as in Figure 15 top), indicating that 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 vs. 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.

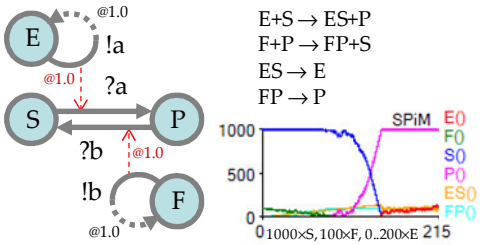

Figure 16 Ultrasensitivity

Therefore we have obtained a device that can compare the relative levels of two slowly varying weak signals (E, F), and produce a strong, quickly-switching output signal that means E > F.

3.6 Positive Feedback Transitions

We now come to another programming exercise in automata collectives. None of the curves that we have seen so far are symmetric; we can then ask: *can we make a symmetric bell shape* as in Figure 18? This question will lead us to building another basic analog component: an oscillator.

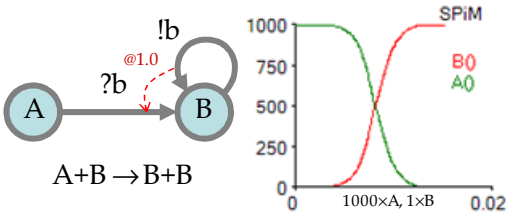

Figure 17 Positive feedback transition

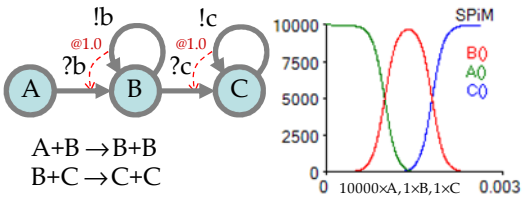

Figure 18 Bell shape

A theorem of probability theory guarantees that we can in fact approximate any shape we want by combining exponential distributions, but the resulting automata would normally be huge. Here we are looking for a compact programming solution. One

way to obtain a sharp raising transition (the first half of a bell curve) is by positive feedback. In Figure 17, the more B's there are, the faster the A's are transformed into B's, so the B's accumulate faster and faster, up to saturation. (Note that at least one B is needed to bootstrap the process.)

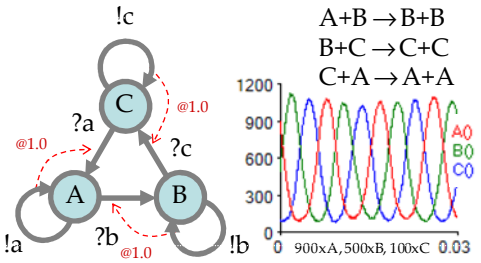

Figure 19 Oscillator

By linking two such transitions in series (Figure 18), we obtain a symmetrical bell shape. We can see that after the B's start accumulating, they are being drained faster and faster by the accumulating C's. The fact that the B's are being drained in a symmetrical way can be explained by the kinetics of B: $[B]^{\bullet} = [B]([A]-[C])$. (Note that there is a very small chance that the B's will be drained by the C's before the wave can accumulate in B, therefore stalling it.)

Linking several such transitions in series (not shown) produces a soliton-like wave that does not dissipate (a delay transition between adjacent states is needed to prevent stalling).

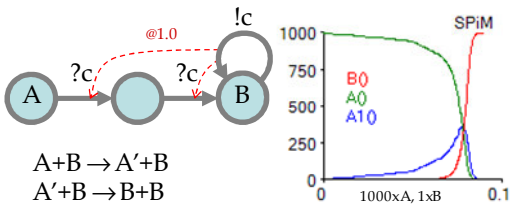

Figure 20 Positive two-stage feedback

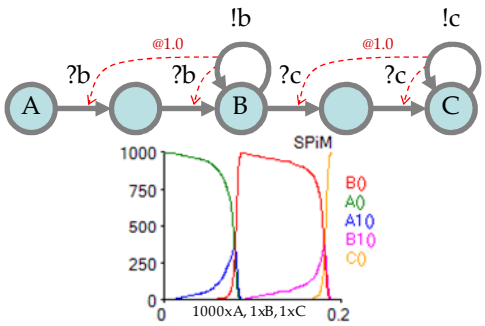

Figure 21 Square shape

Linking three positive feedback transitions in a loop produces a stochastic oscillator (Figure 19); moreover, the ODEs extracted from the chemical reactions describe a deterministic (never stopping) oscillator. A sustained stochastic 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; when that happens (usually after a long time) there is nothing to pull on the next wave, and the oscillation stops.

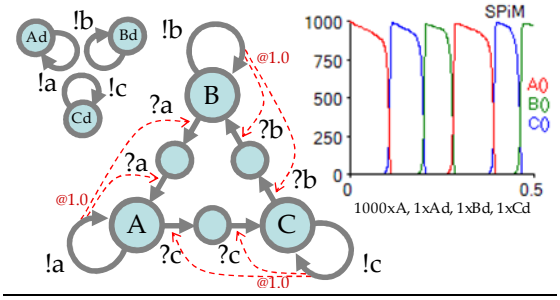

Figure 22 Hysteric 3-way groupies

An interesting variation is a two-stage positive feedback loop (Figure 20) where the drop of state A is delayed and the growth of state B is steeper. Joining two such transitions (Figure 21) produces a shape that approximates a rectangular wave as we increase the cardinality of A. Linking three such transitions in a loop produces again an oscillator (Figure 22). 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. This oscillator is a 3-states version of the oscillators in Figure 7, and is very robust.

3.7 Excitation Cascades

Beyond signal comparators and oscillators, other basic analog devices we may want to build include signal amplifiers and dividers. We next imitate some amplifiers found in biological systems that are made of cascades of simpler stages. As a technical note, the modular nature of these staged amplifiers leads us to write compact, parametric simulation code that is instantiated at each stage. This compactness is not reflected in the figures, where we simply redraw each stage. But, as a consequence, it is more convenient in the simulation code to plot the counts of active outputs, e.g., !a, !b, !c, rather than the counts of the states that produce those outputs, e.g. aHi, bHi, cHi: this plotting style is adopted from now on.

We consider cascades where one enzyme activates another enzyme. A typical situation is shown in Figure 23 (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. This network can be considered as the skeleton of a

MAPK cascade, which similarly functions as an amplifier with three stages of activation, but which is more complex in structure and detail [10].

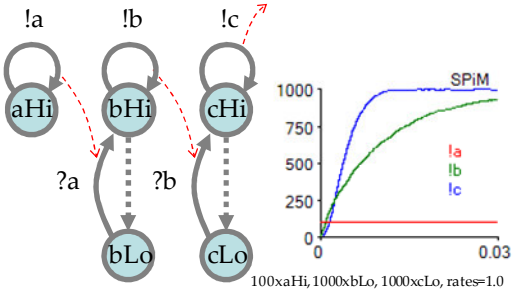

Figure 23 Second-order cascade

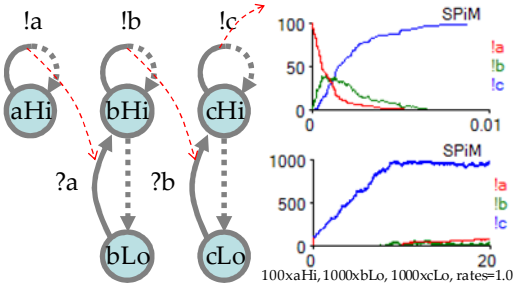

Figure 24 Zero-order cascade

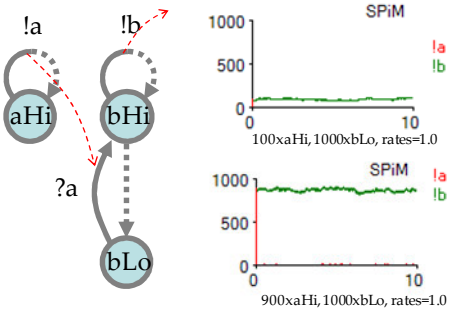

Figure 25 Zero-order cascade, 1 stage

The resulting amplification behavior, however, is non-obvious, as can be seen by comparison with the zero-order activation cascade in Figure 24; the only difference there 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 [21], but apparently can be obtained also by zero-order kinetics. Of course, the behavior in Figure 23 is the limit of that in Figure 24, as we decrease the zero-order delay.

A single stage of a second-order cascade works like the bHi level shown in Figure 23 (since no bHi is actually consumed by the next stage), that is, as an amplifier. Surprisingly, a single stage of the zero-order cascade, works quite differently, as a signal replicator. In Figure 25, a given level of aHi (=100 or =900), induces an equal level of bHi, as long as it is lower than the reservoir of bLo (=1000). (If aHi exceeds bLo, then bHi:=bLo and aHi:=aHi-bLo.) However, the two-stage cascade in Figure 24 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.

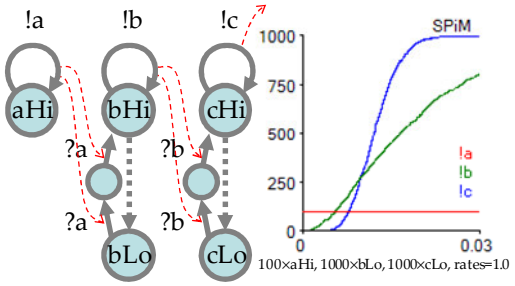

Figure 26 Second-order double cascade

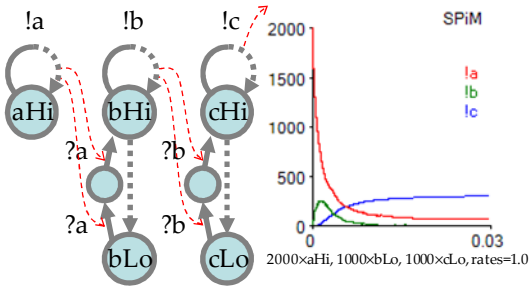

Figure 27 Zero-order double cascade

Real MAPK cascades are actually based on double activation, as shown in Figure 26, where the sigmoid output is more pronounced and delayed than in Figure 23. And once again, the zero order regime brings surprises: the cascade in Figure 27 works in reverse, as a signal attenuator, where even a very high amount of aHi produces a low stable level of cHi which is at most 1/3 of maximum cHi. This is because one stage of such a cascade is actually a signal level divider, where if bHi=1000 then cHi=333, with the signal being distributed among the three states of the stage.

3.8 Boolean Inverters

Having seen how to obtain basic analog functions (comparators, oscillators, amplifiers, and dividers) it is now time to consider digital devices: inverters, and other Boolean gates. Automata with distinguished "low" 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 28: 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. (Steady state analysis of the ODEs shows that [bHi] = max/(1+γ[aHi]), where max = [bHi]+[bLo].)

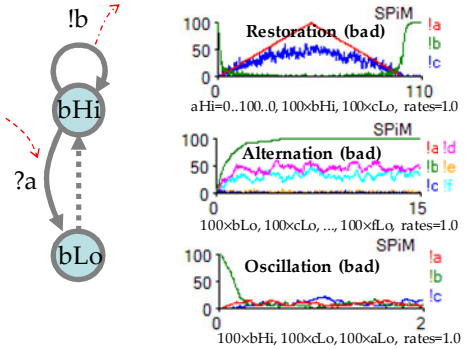

Figure 28 Simple inverter

We test the behavior of populations of inverters according to three quality measures, which are first applied to the inverter in Figure 28:

(1) Restoration. With two inverter populations in series (100×Inv(a,b) + 100×Inv(b,c)), a triangularly-shaped input signal (?a) is provided that ramps up from 0 to 100 and then back down to 0.

(2) Alternation. We test a connected sequence 100×Inv(a,b) + 100×Inv(b,c) + … + 100×Inv(e,f).

(3) Oscillation. We test a cycle of three populations, 100×Inv(a,b) + 100×Inv(b,c) + 100×Inv(c,a).

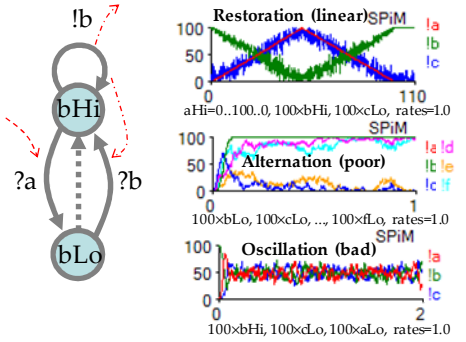

Figure 29 Feedback inverter

In Figure 28, top plot, we see that the first stage 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 second stage !c output is neither a faithful reproduction nor a Boolean restoration of the ?a input. In the middle plot, we see that intermediate signals in the alternation sequence are neither high nor low: the Boolean character is lost. In the bottom plot, we see that three gates in a loop fail to sustain a Boolean oscillation. Therefore, we conclude that this inverter does not have good Boolean characteristics, possibly because it reacts too strongly to a very small input level, instead of switching on a substantial signal.

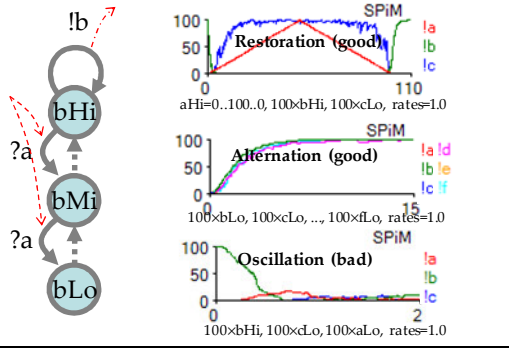

Figure 30 Double-height inverter

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 (Figure 29). The result is, unexpectedly, a linear signal inverter. (We can deduce the linearity from the steady state analysis of the ODEs: [bLo] = [aHi][bHi]/([bHi]+1/γ) ≈ [aHi] for [bHi]>>1/γ, 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.

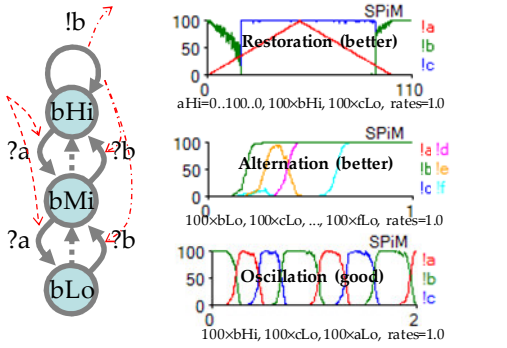

Figure 31 Double-height feedback inverter

A good Boolean inverter can be obtained, instead, by doubling the height of the simple inverter (Figure 30). This double height inverter gives perfect alternation, and good restoration (transforming a triangular input, !a into a nearly rectangular output, !c). However, it still fails to oscillate.

Finally, we combine the two techniques in a double-height feedback inverter (Figure 31). This has perfect restoration, transforming a triangular input into a sharp rectangle. It also has strong and quickly achieved alternation, and regular oscillation.

In conclusion, it is possible to build good Boolean inverters and signal restorers. This is important because it lessens the requirements on other circuits: if those circuits degrade signals, we can always re-

store a proper Boolean signal by two inverters in series. We examine some Boolean circuits next.

3.9 Boolean Circuits

It probably seems obvious that we can build Boolean circuits out of populations of automata, and hence support general computation. However, it is actually surprising, because finite populations of interacting automata are *not* capable of general computation [2], and only by using stochasticity one can approximate general computation up to an arbitrarily small error [22]. Those are recent results, and the relationship with the Boolean circuits shown here is not clear; likely one needs to use sufficiently large populations to reduce computation errors below a certain level.

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

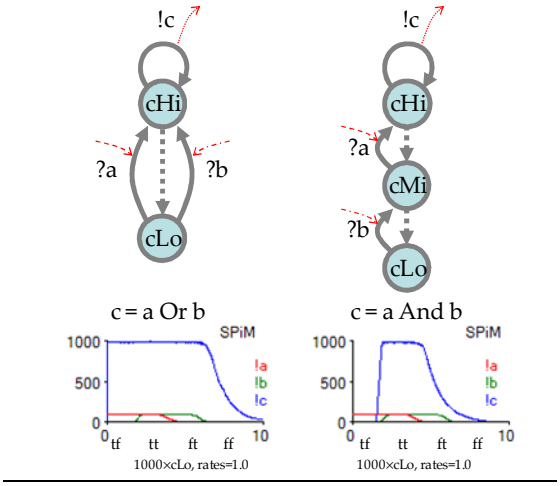

Figure 32 Or and And

Figure 32 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).

The chemical reactions for the Or gate are $aHi+cLo \to^{\gamma} aHi+cHi$, $bHi+cLo \to^{\gamma} bHi+cHi$, $cHi \to cLo$. From their ODEs, by setting derivatives to zero, and with the automata constraint [cHi]+[cLo] = max, we obtain [cHi] = max([aHi]+[bHi])/([aHi]+[bHi]+1/γ). That is, if the inputs are zero then [cHi] = 0. If, say, [aHi] is non-zero, then [cHi] = max[aHi]/([aHi]+1/γ) so that for [aHi] >> 1/γ we have [cHi] ≈ max.

The And gate can be similarly analyzed. Note that And is not perfectly commutative because the decay back to cLo on a single input is slightly different depending on which input is provided. However, as usual, any analog implementation of a digital gate must be given enough time to stabilize.

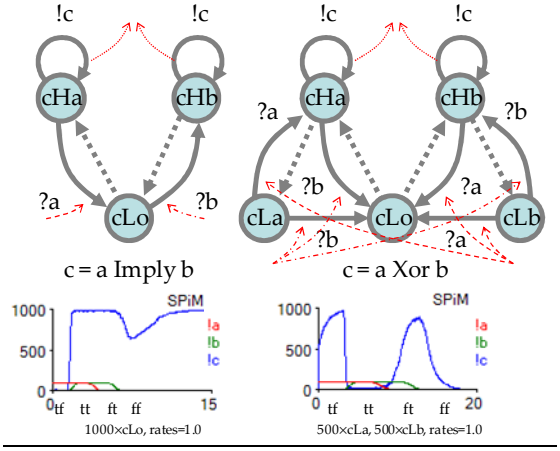

Figure 33 Imply and Xor

Figure 33 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 cHb to the high state cHa, through cLo, in a transition from *false* Imply *true* to *false* Imply *false*.

The steady state behavior of Imply is: output = [cHa] + [cHb] = max - max[aHi] / ([aHi][bHi] + [aHi] + 1/γ) where max is the size of the collective. If [aHi] = 0 we have output = max; if [aHi] ≠ 0 and [bHi] = 0 we have output ≈ 0; if [aHi] ≈ [bHi] ≈ max, we have output ≈ max.

Although Xor can be constructed from a network of simpler gates, the Xor gate in Figure 33 is implemented as a single uniform collective.

3.10 Bistable Circuits

We have now assembled a collection of basic analog and digital devices that can be used to perform signal processing and combinatorial computation. For completeness, we need to discuss also memory elements; these can be constructed either as bistable digital circuits (flip-flops), using the gates of Sections 3.8 and 3.9, or as bistable analog devices.

We have already seen examples of bistable analog devices: the ultrasensitive comparator of Section 3.5, and the groupies of Section 2.2. The groupies, however, are not stable under perturbations: any perturbation that moves them away from one of the two stable states can easily cause them to wander

into the other stable state. Hence, they would not be very good as stable memory elements.

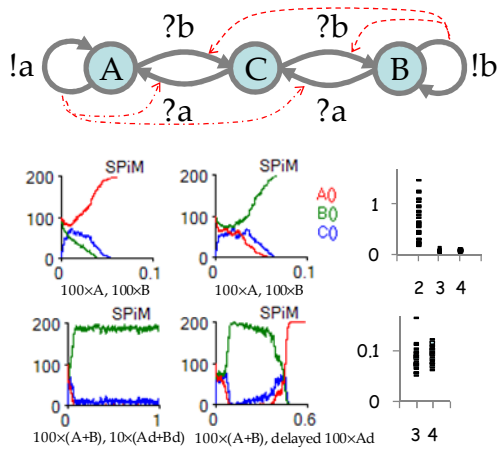

Figure 34 Memory Elements

In Figure 34 we show a modified version of the groupies, obtained by adding an intermediate state shared by the two state transitions. This automaton has very good memory properties. The top-left and top-center plots show that it is in fact spontaneously bistable. The bottom-left plot shows that it is stable in presence of sustained 10% fluctuations produced by doping automata. The bottom-center plot shows that, although resistant to perturbations, it can be switched from one state to another by a signal of the same magnitude as the stability level: the switching time is comparable to the stabilization time. In addition, this circuit reaches stability 10 times faster than the original groupies: the top-right plot shows the convergence times of 30 runs each of the original groupies with 2 states, the current automaton with 3 states, and a similar automaton (not shown) with 4 states that has two middle states in series. The bottom-right plot is a detailed view of the same data, showing that the automaton with 4 states is not significantly faster than the one with 3 states. Therefore, we have a stable and fast memory element.

3.11 Discrete vs. Continuous Modeling

We finally get back to the oscillating behavior of the hysteric groupies of Figure 7. In the previous sections we have analyzed many systems both by stochastic simulation and by differential equations, implying that in most cases we have a match between the two approaches, and that we can use whatever analysis is most useful. However, continuous techniques are not always appropriate [24][1], and there are well-known examples of that [25].

As an illustration of the general issue, in Figure 35 we compare the stochastic simulation of groupie collectives against numerical solutions of their corresponding differential equations (found in the Figure 35 scripts in Appendix). On the left column we have the basic groupies from Figure 5, and on the middle and right columns we have the groupies with one or two intermediate steps from Figure 7; in all cases we include a low number of doping automata, as in Figure 7, to prevent deadlocks. The bottom row has plots of the stochastic simulations, all starting with 2000 automata in state A. (Comparable, noisier, simulations with 200 automata are found in the mentioned figures.) The top row has instead the ODE simulations with the same initial conditions (including doping), and with the volume of the solution taken as γ=1.0.

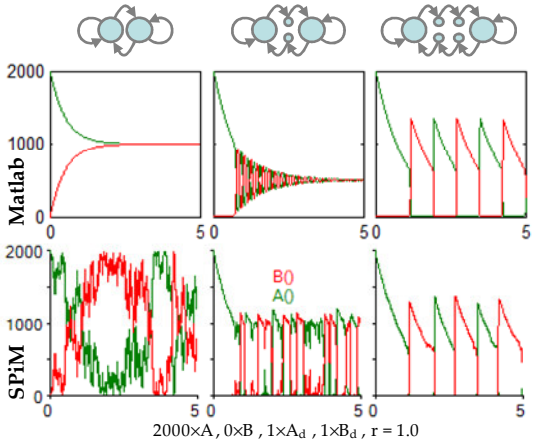

Figure 35 Discrete vs. continuous modeling

As we can see, in the right column there is an excellent match between the stochastic and deterministic simulations; moreover, the match gets better when increasing the number of molecules, as expected. In the middle column, however, after a common initial transient, the deterministic simulation produces a dampened oscillation, implying that all 4 states are eventually equally occupied. Instead, the stochastic simulation produces an irregular but not at all dampened oscillation, where the A and B states persistently peak at twice the continuous values (detailed plots reveal that A and its successor state peak together, and so do B and its successor). Even more strikingly, in the left column, the deterministic simulation produces an equilibrium level, which also happens to be stable to perturbations, while the stochastic simulation produces a completely unstable random walk.

The conclusions one should draw from this situation is that it is not always appropriate to use a deterministic approximation of a stochastic system (and it is likely impossible in general to tell when it is appropriate). The difference can be particularly

troublesome when studying just one run of one copy of a stochastic system (e.g. the behavior of one cell), as opposed to the average of a number of runs, or the average of a number of copies. Unfortunately, to understand the behavior of, e.g., cells, one must really study them one at a time. Stochastic behavior can be characterized precisely by other kinds of differential equations (the Chemical Master Equation), which are always consistent with the stochastic simulations, but those are much more difficult to analyze than the mass action ODEs [5].

4 The Biochemistry of Automata

4.1 Beyond simple automata

As we have seen, the simple automata of Section 2 can model typical chemical interactions. Biochemistry, however, is based on a richer set of molecular interactions, and in this section we explore a corresponding richer notion of interacting automata.

A characteristic feature of biochemistry, and of proteins in particular, is that biological molecules can stick to each other to form *complexes*. They can later break up into the original components, with each molecule preserving its identity. 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 identical molecules (*monomers*): chemically this can be represented only by an unbounded number of chemical species, one for each length of a polymer, which is obviously cumbersome and technically infinite.

In order to model the complexation features of biochemistry directly, we introduce *polyautomata*, which are automata that can form reversible complexes, 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 it first dissociates. Association and dissociation can be encoded in π-calculus [14] (as shown in Appendix: Figure 36), by taking advantage of its most powerful features: fresh channels and scope extrusion. That encoding results in flexible modeling of complexation [19], but does not enforce constraints on reassociation.

Here we strive again to remain within the confines of an automata-like framework, including diagrammatic descriptions. A formal presentation of polyautomata is given in [2], where it is shown that they are Turing-complete in conjunction with an unbounded supply of a finite number of monomer species (and that instead, the automata of Section 2 can achieve Turing-completeness only with an unbounded supply of *different* species, which means infinite-size programs).

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 *current associations*.

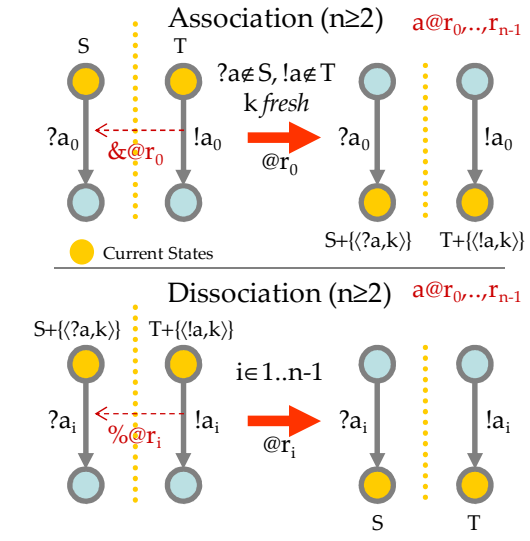

Figure 36 Polyautomata reactions

An association is a pair ⟨π,k⟩ where π is an event label (?a or !a for the complementary sides of an association), 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 multiple ways of disassociating two automata after a given association; the rates of association and of each possible disassociation can differ. Therefore, each channel will now be 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.

If arity(a)=1, then r_0 is called the *interaction rate*, because it covers the old case of ordinary interac-

tions: 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 $r_1,\ldots,r_{n-1}$ are the *dissociation rates*. The association rules from Figure 36 then apply.

An association (Figure 36 top) on a channel *cannot* happen if an automaton has a past association event 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 preconditions $?a \notin S$ (short for $\langle ?a,k\rangle \notin S$ for any k) and $!a \notin T$, check for such conflicts, 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. In examples we use instead the notation &?a and &!a for these labels, where & indicates association, omitting index 0.

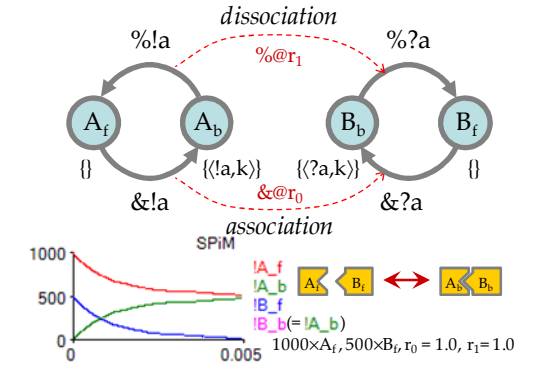

Figure 37 Complexation/decomplexation

Symmetrically, a dissociation (Figure 36 bottom) on a channel happens only if the two automata have a current 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. 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 37 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 yellow cartoon shapes illustrates the mechanics of complexation, where complexation channels are depicted as complementary surfaces. The plot shows that for the chosen rates, the dynamic equilibrium is heavily biased towards the bound states. (In this section a state A_f corresponds to the plot line !A_f.)

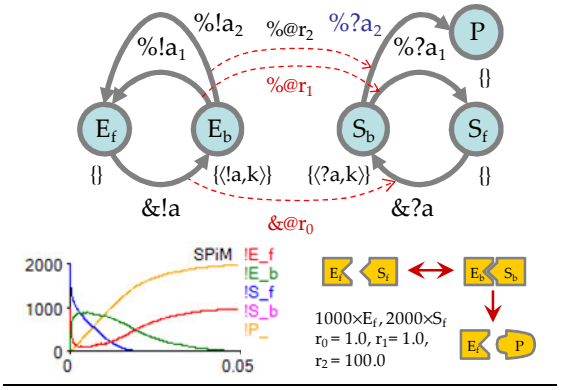

Figure 38 Enzymatic reactions

The use of multiple dissociation rates is exemplified by enzymatic reactions (Figure 38): these are now the true enzymatic reactions, not the ones from Section 3.4 [19]. From the bound state of enzyme (E_b) and substrate (S_b), two dissociations are possible with the one at higher rate producing product (P).

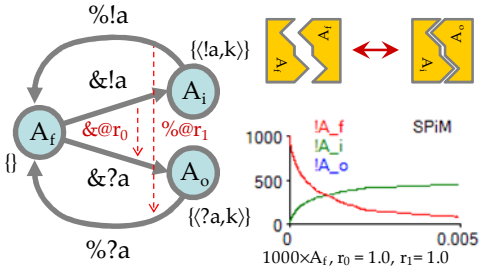

Figure 39 Homodimerization

More subtle forms of complexation exist. Homodimerization (Figure 39) is symmetric complexation: a monomer offers both an input and an output complexation on the same channel, meaning that it offers two complementary surfaces, and can stick to a copy of itself. Note that a monomer here cannot bind to two other monomers over its two complementary surfaces. That situation leads to polymerization, as shown next.

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 linear ones.

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.

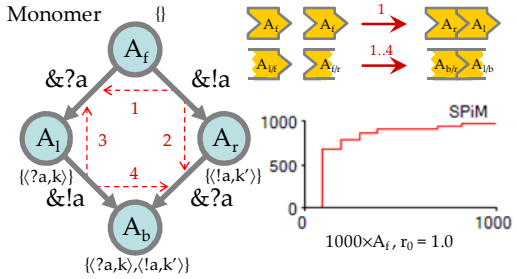

Figure 40 Bidirectional polymerization

In Figure 40 we show a monomer automaton that 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, with two association events). The sequence of transitions is 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 41 is a schematic representation of some possible configurations of monomers, with thick lines joining their current states and representing their current associations.

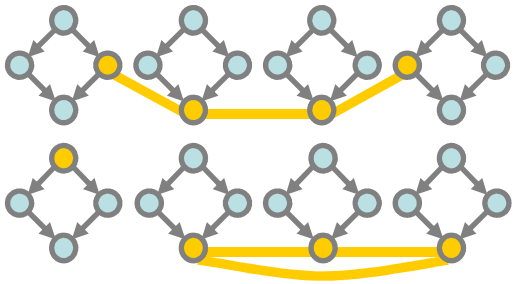

Figure 41 Automata polymers

The plot in Figure 40 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 different form of polymerization, inspired by the actin biopolymer, which can grow only at one end and shrink only at the other end. In Figure 42 we have the same four monomer states, but the sequencing of transitions is different.

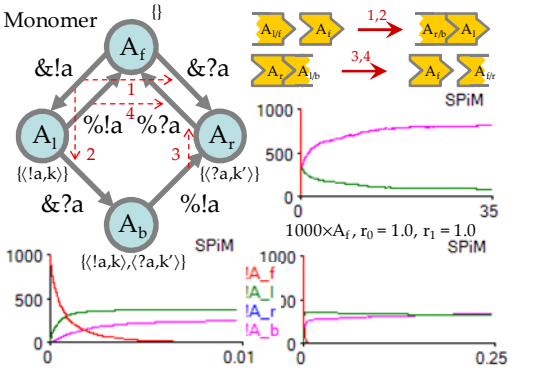

Figure 42 Actin-like polymerization

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. 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 42 show three views of the same simulation run with 1000 monomers, at times 0.01, 0.25, and 35; 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 43 shows in detail 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 thick lines indicate the current associations, which are actually encoded in the association sets shown under the current states, by the shared k and j.

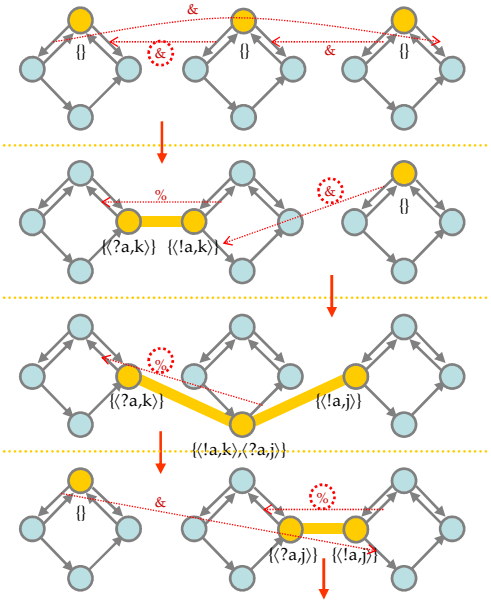

Figure 43 Typical monomer interactions

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.

In conclusion, polyautomata provide a relatively simple graphical notation for representing combinatorial systems of interacting *and* complexing molecules. Systems that grow without bounds by complexation can be represented compactly and finitely: this would not be possible using the automata of Section 2 or, in fact, using chemical reactions.

5 Conclusions and Related Work

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 [11]. *Formal notations* (such as programming languages and process algebras) are fundamental tools for achieving those goals: they enable engineering and analysis techniques that are orthogonal to the ones available by *mathematical models* (such as set theory, calculus, and Markov chains). In computing, adequate notation is key to the maintainability of large information processing systems consisting of millions of lines of code, whose complexity is dwarfed only by biological systems. Noticeably, information processing systems are not written using differential equations, nor set theory, because those are not useful tools in that domain.

Still, it is always important to relate formal notation to mathematical models. We have used automata notation for exploring simple but intriguing biochemical systems, aiming to demonstrate how easy it is to "play with" the notation to get insights into a system. We have shown, by example, how to relate the interacting automata notation to stochastic behavior and to differential equations. For a full development, the process algebra foundations for this work are found in [1], which connects the stochastic π-calculus approach to modeling biochemistry [19], to stochastic and deterministic chemical kinetics [24]. Additionally, foundations for Section 4 are found in [2].

The use of compositional, graphical, formal notation has been long advocated [6], but most graphical notations in systems biology still lack such fundamental properties. Our automata are at least compositional, but are neither parameterizable nor scalable, unless they are embedded in the richer framework of process algebras, which has all such properties. Our diagrams are based on previous work on the graphical representation of the whole π-calculus [16]. Dealing with the full π-calculus however means having to graphically represent, in general, bound variables. The full π-calculus also seems excessive for use in biochemical models, both semantically and graphically. Therefore, while the diagrams in Sections 1-3 are essentially a reduced version of the ones in [16], the diagrams in Section 4 opt to use operators instead of bound variables to deal with the important biochemical operation of complexation, and enforce also an invariant against reassociation of occupied sites.

The pragmatics of graphical and formal notation for the biochemical domain still requires investigation. It has taken decades to develop adequate notations and analysis techniques for large software and hardware systems; we are just at the beginning to do the same for biochemical systems, where the task will certainly be much harder.

Acknowledgments

Much of this material was initially prepared for a doctorate course of the same title at the University of Trento, in May 2006, and improved by referees.

References

[1] L. Cardelli: On process rate semantics. Theoretical Computer Science, 391(3) 190-215. Elsevier, 2008. <http://dx.doi.org/10.1016/j.tcs.2007.11.012>

[2] L. Cardelli, G. Zavattaro. On the Computational Power of Biochemistry. To appear. <http://lucacardelli.name/Papers/On the Computational Power of Biochemistry.pdf>

[3] 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.

[4] D.T. Gillespie. Exact Stochastic Simulation of Coupled Chemical Reactions. Journal of Physical Chemistry 81, 2340–2361. 1977.

[5] T. Jahnke, W. Huisinga. Solving the chemical master equation for monomolecular reaction systems analytically. Mathematical Biology 54(1): 1-26. Jan 2007.

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

[7] D. Harel, S. Efroni and I. R. Cohen, "Reactive Animation", Proc. FMCO 2002. LNCS 2852:136-153, Springer, 2003.

[8] J.Hillston. A compositional approach to performance modelling. Cambridge University Press, 1996.

[9] J. E. House: Principles of Chemical Kinetics. Academic Press, 2007.

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

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

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

[13] 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.

[14] Milner, R.: Communicating and mobile systems: the π-Calculus. Cambridge University Press, 1999.

[15] A. Phillips, L. Cardelli: A correct abstract machine for the stochastic pi-calculus. Proc. BioConcur'04.

[16] A. Phillips, L. Cardelli, G. Castagna: A graphical representation for biological processes in the stochastic pi-calculus. Transactions on Computational Systems Biology VII, 123-152. Springer, 2006.

[17] C. Priami. Stochastic π-calculus. The Computer Journal, 38, pp 578–589, 1995.

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

[19] A. Regev, E. Shapiro: The π-calculus as an Abstraction for Biomolecular Systems. In G. Ciobanu, G. Rozenberg (Eds.): Modelling in Molecular Biology, 219-266, Springer, 2004..

[20] A. Regev, E. Shapiro. Cellular abstractions: Cells as computation. Nature 419 343. 2002.

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

[22] D. Soloveichik, M. Cook, E. Winfree, and J. Bruck. Computation with Finite Stochastic Chemical Reaction Networks, 2007. <http://www.dna.caltech.edu/DNAresearch publications.html>.

[23] K. Tumer, D. Wolpert: A survey of collectives. In Collectives and the Design of Complex Systems, 1-42. Springer, 2004.

[24] O. Wolkenhauer, M. Ullah, W. Kolch, K. Cho. Modelling and simulation of intracellular dynamics: choosing an appropriate framework. IEEE Transactions on NanoBioscience 3, 200-207. 2004.

[25] J. M. G. Vilar, H. Y. Kueh, N. Barkai, S. Leibler. Mechanisms of noise-resistance in genetic oscillators. PNAS 99(9) 5988-5992. 2002.

Appendix: Simulations

These scripts are for the SPiM Player stochastic π-calculus simulator v1.13 [15], and Matlab 7.4.0 (ode45). The core SPiM syntax maps directly to stochastic π-calculus [17][18]. The SPiM scripts are complete and executable, and usually are a literal translation of the automata in the figures, with some additional code for plotting directives and for test signals. Figure 2 and Figure 36 instead outline the encoding of automata used in the other scripts.

Figure 2: Automata reactions

SPiM encoding of Delay at rate r from state A to state B, then running 100 automata with initial state A:

```
let A() = delay@r; B()   and B() = ...
run 100 of A()
```

Encoding of Interaction: an input ?a from state A1 to A2 and an ouput !a from state B1 to B2, over a channel of rate r, between two concurrent automata initially in states A1 and B1.

```
new a@r:chan
let A1() = ?a; A2()   and A2() = ...
let B1() = !a; B2()   and B2() = ...
run (A1() | B1())
```

SPiM encoding of multiple transitions (a delay, an input, and an output) from the same state A to three different states:

```
let A() = do delay@r; B1() or ?a; B2() or !b; B3()
```

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 5: 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 6: 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 7: 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 9: 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 11: 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 12: Same behavior

```
directive sample 0.02
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 1000 of  B()
```

Figure 13: Sequence of interactions

```
directive sample 0.02
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 5000 of (A1() | B1())
```

Figure 14: 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 15: Subtraction

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

new a@1.0:chan

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

let raising(p:proc(), t:float) =
(* Produce one p() every t sec with precision dt *)
  (val dt= 100.0 run step(p, t, dt, dt))
and step(p:proc(), t:float, n:float, dt:float) =
   if n<=0.0 then (p()|step(p,t,dt,dt))
   else delay@dt/t; step(p,t,n-1.0,dt)

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 15

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

Figure 16: Ultrasensitivity

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

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

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

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

run 1000 of S()

let raising(p:proc(), t:float) =
  ... see code for Figure 15

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

Figure 17: 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 18: 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 19: Oscillator

```
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 20: Positive two-stage feedback

```
directive sample 0.1 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 21: 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 22: 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 23: 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 24: 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 25: Zero-order transduction

```
directive sample 10.0
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 900 of A()
```

Figure 26: 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 27: 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 28: 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) =
  (val dt=100.0 run step(tick, t, dt, dt))
and step(tick:chan, t:float, n:float, dt:float) =
  if n<=0.0 then !tick; clock(t,tick)
  else delay@dt/t; step(tick,t,n-1.0,dt)
let S1(a:chan, tock:chan) =
  do !a; S1(a,tock) or ?tock; ()
and 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 29: 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 28

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 ... and Inv_lo ...
    ... see code above

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 ... and Inv_lo ...
    ... see code above

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

Figure 30: 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 28

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 ... and Inv2_lo ...
    ... see code above

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 ... and Inv2_lo ...
    ... see code above

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

Figure 31: 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 28

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 ... and Inv2_lo ...
    ... see code above

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 ... and Inv2_lo ...
    ... see code above

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

Figure 32: 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) =
  (val dt=100.0 run step(tick, t, dt, dt))
and step(tick:chan, t:float, n:float, dt:float) =
  if n<=0.0 then !tick; clock(t,tick)
  else delay@dt/t; step(tick,t,n-1.0,dt)


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

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

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 32

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

Figure 33: 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 32

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 32

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

Figure 34: Memory Elements

```
(* Top Left, Top Center *)
directive sample 0.1
directive plot A(); B(); C()

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

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

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

```
(* Bottom Left *)
directive sample 1.0
directive plot A(); B(); C()

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

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

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

run 100 of (A() | B())
run 10 of (Ad() | Bd())
```

```
(* Bottom Center *)
directive sample 0.6
directive plot A(); B(); C()

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

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

let Ad() = !a; Ad()

run 100 of (A() | B())
run 100 of delay@10.0; delay@10.0; delay@10.0;
           delay@10.0; delay@10.0;  Ad()
```

Figure 35: Discrete vs. Continuous Modeling

```
(* Top Left *)                 initially
(A)  dx1/dt = -(x1-x2)         x1 = 2000.0
(B)  dx2/dt = (x1-x2)          x2 = 0.0
```

```
(* Top Center *)               initially
(A)  dx1/dt=x1*x4-x3*x1-x1+x4  x1 = 2000.0
(A') dx2/dt=x3*x1-x3*x2+x1-x2  x2 = 0.0
(B)  dx3/dt=x3*x2-x1*x3-x3+x2  x3 = 0.0
(B') dx4/dt=x1*x3-x1*x4+x3-x4  x4 = 0.0
```

```
(* Top Right *)                initially
(A)  dx1/dt=x1*x6-x3*x1-x1+x6  x1 = 2000.0
(A') dx2/dt=x3*x1-x3*x2+x1-x2  x2 = 0.0
(A") dx5/dt=x3*x2-x3*x5+x2-x5  x5 = 0.0
(B)  dx3/dt=x3*x5-x1*x3-x3+x5  x3 = 0.0
(B') dx4/dt=x1*x3-x1*x4+x3-x4  x4 = 0.0
(B") dx6/dt=x1*x4-x1*x6+x4-x6  x6 = 0.0
```

```
(* Bottom Left *)
directive sample 5.0 1000
directive plot B(); A()

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()

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

```
run 2000 of A()
run 1 of (Ad() | Bd())
```

```
(* Bottom Center *)
Same as Bottom Left, except:
let A() = do !a; A() or ?b; ?b; B()
and B() = do !b; B() or ?a; ?a; A()
```

```
(* Bottom Right *)
Same as Bottom Left, except:
let A() = do !a; A() or ?b; ?b; ?b; B()
and B() = do !b; B() or ?a; ?a; ?a; A()
```

Figure 36: Polyautomata reactions

SPiM encoding of Association over channel $a@r_0,r_1$ of arity 1, with one automaton performing an output from state A to A1 and the other automaton performing an input from state B to B1:

```
new a@r0:chan(chan)
let A() = (new k1@r1:chan run !a(k1); A1(k1))
and B() = ?a(k1); B1(k1)
```

Encoding of Dissociation through the previously shared k1.

```
and A1(k1:chan) = !k1; A()
and B1(k1:chan) = ?k1; B()
```

More generally, for $a@r_0,...,r_{n-1}$ we declare an (n-1)-ary channel:

```
new a@r0:chan(chan,...,chan)      (*n-1 times*)
```

Association then creates n-1 shared dissociation channels:

```
let A() = (new k1@r1:chan ... new kn-1@rn-1:chan
           run !a(k1,...,kn); A1(k1,...,kn))
```

and then A1 can choose which channel to use for dissociation. Note that the constraint about not reassociating before a dissociation is not automatically enforced by this encoding.

Figure 37: Complexation/decomplexation

```
directive sample 0.005
directive plot !A_f; !A_b; !B_f; !B_b
new A_f:chan new A_b:chan new B_f:chan new B_b:chan

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

let Af() = (new k@lam:chan run do !a(k); Ab(k) or !A_f)
and Ab(k:chan) = do !k; Af() or !A_b

let Bf() = do ?a(k); Bb(k) or !B_f
and Bb(k:chan) = do ?k; Bf() or !B_b

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

Figure 38: Enzymatic reactions

```
directive sample 0.05 1000
directive plot !E_f; !E_b; !S_f; !S_b; !P_
new E_f:chan new E_b:chan
new S_f:chan new S_b:chan new P_:chan

val r0 = 1.0    val r1 = 1.0    val r2 = 100.0
new a@r0:chan(chan,chan)

let P() = !P_

let Ef() =
   (new k1@r1:chan new k2@r2:chan
    run do !a(k1,k2); Eb(k1,k2) or !E_f)
and Eb(k1:chan,k2:chan) =
  do !k1; Ef() or !k2; Ef() or !E_b

let Sf() = do ?a(k1,k2); Sb(k1,k2) or !S_f
and Sb(k1:chan,k2:chan) =
  do ?k1; Sf() or ?k2; P() or !S_b

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

Figure 39: Homodimerization

```
directive sample 0.005 10000
directive plot !A_f; !A_i; !A_o
new A_f:chan new A_i:chan new A_o:chan

new a@1.0:chan(chan)

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

and Ai(k:chan) = do ?k; Af() or !A_i
and Ao(k:chan) = do !k; Af() or !A_o

run 1000 of Af()
```

Figure 40: Bidirectional polymerization

```
directive sample 1000.0
directive plot ?count

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) =
  (val dt=100.0 run step(tick, t, dt, dt))
and step(tick:chan, t:float, n:float, dt:float) =
  if n<=0.0 then !tick; clock(t,tick) else delay@dt/t;
step(tick,t,n-1.0,dt)

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

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

Figure 42: Actin-like polymerization

```
directive sample 0.01 (* 0.25, 35.0 *) 1000
directive plot !A_f; !A_l; !A_r; !A_b
new A_f:chan new A_l:chan new A_r:chan new A_b:chan

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) or !A_f)

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

and Ar(rht:chan) =
   do ?rht; Af() or !A_r

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

run 1000 of Af()
```