A Model of Organizational Closure, Formal Languages for Autonomy

Ian Wood, Project Advisor: Randall Beer, Ph.D.

February 12, 2015

In the autopoietic definition of life there are many concepts that are informally defined, resulting in a ambiguities that formal systems can help elucidate. One of the most fundamental ambiguities is how processes and components must relate to one another in an autopoietic system. In this paper I discuss the concepts involved in autopoiesis and organizational closure as defined by Maturana and Varela, describe some minimal models that clarify some of these concepts, and finally focus on a lambda-calculus based model that makes explicit its assumptions behind the duality between component and process.

1 Introduction

An autopoietic system is defined:

> As a network of processes of production (transformation and destruction) of components that: (1) through their interactions and transformations continuously regenerate and realize the network of processes (relations) that produced them; and (2) constitute it (the machine) as a concrete unity in the space in which they exist by specifying the topological domain of its realization as such a network. (Varela 1979)

This definition captures a basic intuition about a living organism: that it is some invariant beyond the physical components that constitute it. However, this definition and many of the explanations surrounding it leave ambiguous the duality between process and component. How do the components of a system regenerate and realize a network of processes? How much of these processes should be contributed to the components, and how much to the observer and his reference to a whole physics in which the components exist? To what extent is such an operationally closed system autonomous if its description relies on physical laws that super-cede its components by specifying when interactions may occur between them?

Maturana clarifies some aspects of these questions in his definitions of unities, but to define unities we must also define the observer. As Maturana points out "Everything

said is said by an observer" (Maturana 1980), and everything discussed is ultimately in reference to the praxis of living of the observer, or the phenomena that the observer can distinguish (and discuss). Distinction is the basic act of the observer, and by making a distinction the observer brings forth a unity simultaneously with its medium. A simple unity is then described by the collection of its interactions within its medium, and a composite unity is a unity composed of components acting as simple unities. The composite unity can then be distinguished as a simple unity in a meta-domain distinct from the domain of its components. The composite unity as simple unity has properties determined by the interactions of its components. Composite unities have an additional feature, a duality between structure and organization, where organization refers to the relations between components that make the composite unity a particular kind in its meta-domain, while structure refers to the actual relations between actual components in a particular instance. A composite unity is structurally determined, in that interactions with its external world are specified by the structure of its components and their properties, how they interact with structural configurations of the medium. In this way the composite unity specifies the domains of structural change in both itself and the environment in which it can maintain its organization, otherwise it disintegrates (Maturana 1988). Indeed Maturana and Varela specify that all unities that scientists can deal with are structurally determined, autopoietic or not (Maturana & Varela 1987).

An autonomous system without a topological boundary is a unity defined by an organization that is closed:

> That is, their organization is characterized by processes such that (1) the processes are related as a network, so that they recursively depend on each other in the generation and realization of the processes themselves, and (2) they constitute the system as a unity recognizable in the space (domain) in which the processes exist. (Varela 1979)

Both autopoeitic and organizationally closed systems subsume all their structural changes towards the maintainence of an identity. The result of the system is the system itself, and this is argued to result in significance (Varela 1997) or intrinsic teleology (Weber & Varela 2002). A detailed discussion of these concepts are beyond the scope of this paper.

The history of structural change in a unity without loss of its organization is called ontogeny. When two or more unities have recurrent interactions that are somewhat stable they are said to have a structual coupling (Maturana & Varela 1987). The domain of deformations, or changes to structure, that an autopoietic unity can undergo without loss of its organization is called it's domain of interactions, those without a loss of its identity is its cognitive domain. The deformations can be seen by an observer as the unity's representation of the deforming agent, while the way it compensates is a description of the unity's world. This domain is bounded and dependent upon the unity's structure at a given time, which is a consequence of its ontogeny and structural coupling (Varela 1979).

2 Other Models

2.1 Tesselation Automata

To make these concepts more concrete, Varela provided a minimal model of autopoiesis in the form of a tessellation automata. In his model three components move randomly about on a grid, catalysts, substrates, and links. The catalyst in the presence of two substrates can compose a link, consuming the substrates but not the catalyst. Each link in the presence of another link can concatenate with that link, to form up to two bonds, through which substrates can move but catalysts cannot. Any link may spontaneously disintegrate into two substrates. An initial grid with substrates and a catalyst, along with an algorithm describing the motion and chances for interactions between the components can produce a closed boundary around the catalyst that can rebuild itself after disintegrations, thus distinguishing the system as a unity with a spatial boundary (Varela 1979).

What here are the components and what are the processes? The components as simple unities should be completely characterized by their interactions with their medium. This includes their interactions with other components, as Varela states, but a complete description also requires their interactions with the grid and their interactions with time in the system. The processes are harder to pin down. Perhaps the network of processes, thus organization, is best thought of as the properties of the components relating to interaction with each other. Each process produces components that can be used in another process. However, this seems more like a set of component properties, one of which, the catalyst, is not produced by the system at all.

When considering the structure of the system, further complications arise. Should the structure be limited to the positions of components and the implied potential processes (relations) at a given time step? Or should all processes carried out during a time step and the resulting component positions be considered the structure of the system at that time? In either case the components interact with time and space, but these interactions are outside of the scope of the closed organization. This is not a problem in regards to the definition of autopoiesis as long as time and position aren't considered "produced" by the algorithm driving the system.

The interaction of a system's components with time and space is somewhat captured by the second part of the definition of autopoiesis, wherein the components specify the topological domain of the unity's realization. The system does create a boundary within which the processes of production can be carried out, so that the structure is a bounded unity. It is hard to see how the instantiation of the boundary can be related back to the more abstract organization of the system. Without the boundary the closed structure of the network of productions is still maintained, and indeed the boundary can and is repaired, so the autopoietic system is not said to disintegrate due to its loss of organization when the boundary is broken. Since the processes of production of components would seem to hold without the boundary, ignoring productions of positions in space and time, how the boundary could be specified in terms of an abstract organization is even less clear. The importance of even unstated spatial interactions in the algorithm

is emphasized by the failure of another researcher, McMullin, to recreate the minimal model. There was an additional "chain-based bond inhibition" interaction that prevents links from concatenating in the presence of a doubly-bonded link. Otherwise other loops or strings of links may form within the boundary and prevent the repair of the boundary or prevent substrates from reaching the catalyst (Thompson 2007).

2.2 Game of Life

Beer has investigated a model that makes more explicit the relationship between component and process. His model uses Conway's Game of Life (GoL), a dynamical system in which space and time are divided discretely and the "physics" of the system is a deterministic update rule. Like Varela's tesselation automata, GoL takes place on a two dimensional grid, in which each cell can have a value of 0 ("dead") or 1 ("alive"), and whose "neighborhood" consists of the surrounding eight cells. The update rule is (conceptually) carried out on all cells simultaneously and can be written as follows: if a cell's neighborhood sums to 3 or sums to 2 and the cell's value is 1, the cell's value on the next time step is 1, otherwise the cell's value on the next time step is 0. This simple rule results in a number of interesting patterns, including "blocks" of four 1 cells in a square, "blinkers" of three 1 cells that oscillate between two configurations, and "gliders" of five 1 cells that travels across the grid. Beer describes each of these patterns as autopoietic. The components of a pattern are its 1-cells, and the surrounding 0-cells, which form a boundary for the unity. The processes are then the production, destruction and maintenance of 1-cells and 0-cells carried out during the update step. Each cell's process depends on the processes that produced the values in the neighborhood during the preceeding time step, forming a connected graph, thus obtaining organizational closure (Beer in press). Not only does this model make explicit the relation between component and process, but also between process and space. However, the closure in the case of the glider is obtained in part because an observer can recognize the equivalence of processes displaced in space after one complete cycle.

The structural changes an autopoietic organization can undergo without loss of identity in interaction with its environment is its cognitive domain. Beer investigates all interactions a glider can undergo with its environment by exhaustively observing all possible states of the cells surrounding the glider's boundaries in all possible configurations of the glider. He finds that the glider can describe six equivalence classes of environmental perturbations, and describes the cognitive domain of the glider as the structural change (in terms of the spatial configuration of components) determined by the response of the glider's structure to a perturbation. Beer goes on to investigate possible ontogenies and structural couplings between glider and environment, since the environment is structurally determined as well. While all environments can't possibly be studied, observations over the possible initial environmental configurations 1 and 2 cells beyond the glider's boundary have shown that the structural coupling of the glider is extremely fragile, and the glider often disintegrates after a few interactions with the environment (Beer 2014). This model serves to illuminate the otherwise vague concepts of ontogeny and structural coupling, as well as a subtle distinction made by Maturana

and Varela. In his investigation, Beer equates the glider's domain of interactions with its cognitive domain. However, the full network of changes between structual configurations is only seen by the observer, thus constitutes the "domain of interactions" of the unity while the perturbation classes that describe how a glider changes its structure is thus the "cognitive domain" of the glider (Varela 1979). While there are only two states internally different to the glider, how it transitions between the states (how it deforms to a pertubation) is its "description" of that perturbation, thus all six perturbation classes belong in the cognitive domain of descriptions that the glider can make, even if they transition between only two internal states. This distinction between domain of interactions and cognitive domain is only a semantic one, and is unimportant as long as we observers, like Beer, make explicit the level of description we are operating on and what our terminology means.

3 Lambda-Calculus Model

Fontana and Buss's artificial chemistries are also models that make explicit the connection between component and process, as well as how the components interact. These models are particularly interesting because they do not treat components only as simple unities. The components themselves are composite unities whose interactions are determined by their structure, while keeping the determination itself explicit. However, these models neglect the spatial or topological component necessary for autopoiesis as defined, therefore I will discuss these models in terms of organizational closure.

In these models Fontana and Buss were interested in an existence problem. Darwinian selection can operate to select the most fit organization, but it assumes that it has variation on which it can operate. Innovation, thus variation, in a system may be formed by the intrinsic properties of underlying components, like in chemistry, or by only random noise, like in mutations of DNA or RNA. Yet, even assuming the random mutation of genetic material, without innovation created by the intrinsic properties of components there can be no further effect. How can a genotype generate a phenotype? The dynamical system must not only run on a network of variables representing the interactions between components, it must generate the network itself. In other words, "the system must construct its own state space" (Fontana & Buss 1994)

Both models introduce an explicit duality between object and function, where every element (unity) in the system is an object that can exist in specfic quantities, and also a function that can act upon other objects. These productions can be described with a generalization of the Lotka-Volterra equations, to allow for productions that are not self-replication (Fontana 1991). This captures two abstract features of chemistry, a constructive ability to generate specfic new elements from the interactions of old elements, and the diversity of equivalence classes in chemistry (two or more reactions can produce the same product). These elements are placed in a "flow reactor" where they operate as a "Turing gas". At each step two elements interact according to some collision rule ϕ that includes pragmatic boundary conditions. If the boundary conditions are met the collision produces an element while another element in the reactor is randomly chosen

to be removed so that the total number of elements remains constant. Otherwise the collision is called elastic, and no element is added or removed. Thus, the two abstract features of chemistry are captured, while conservation laws are ignored, and in this way these artificial chemistries should not be confused with a model of real chemistry (Fontana & Buss 1994).

In the flow reactor, objects that are not produced by collisions will gradually be removed by the dilution flux. Thus, only self-maintaining sets of objects can survive the dynamics of the flow reactor over time. To see these sets, we use the help of a map $\mathbf{m}$ applied to a set $\mathcal{A}_t$ of elements such that

$$\mathcal{A}_{t+1} = \mathbf{m}(\mathcal{A}_t) = \mathcal{A}_t \circ_\phi \mathcal{A}_t = \{i|((\phi)j)k \Rightarrow i, (j,k) \in \mathcal{A}_t \times \mathcal{A}_t\} \quad (1)$$

A set $\mathcal{A}$ such that $lim_{n\to\infty}\mathbf{m}^n(\mathcal{A}) = \mathcal{A}^*$ is called a seeding set of the closure $\mathcal{A}^*$. A self-maintaining set is then a seed of the organization $\mathcal{A}^*$. The "center" of the organization is the smallest such seeding set(s). For a fuller discussion of this map and its consequences see (Fontana and Buss 1994)

We can discuss these ideas directly in the terms of autopoiesis/organizational closure. The organization $\mathcal{A}^*$ is either organizationally closed or empty if there are no productive collisions possible. The structure of an instance $\mathcal{A}_t$ of the organization is the set of interacting elements at time t. Organizational closure is lost when no seeding set exists for the organization. Caveat: some sets that are not subsets of $\mathcal{A}^*$ can converge to it, so it is perhaps better to say when no set exists in the basin of attraction of $\mathcal{A}^*$.

Fontana first implemented this system in "Algorithmic Chemistry" (AlChemy) (Fontana 1991). AlChemy is syntactically easy to form, with only one rule: well-formed strings must have an equal number of left and righ parens. However, it relies on primitive operatives and each function can have only a single variable. Without the ability to form higher-order functions through abstraction, the system is limited, and held to the availability of operatives in the system. The later minimal model for a theory of biology is more powerful and produces more interesting results, so I will focus on it for the remainder of the paper.

The rest of this section is divided as follows: a discussion of Fontana and Buss's minimal theory of biology, my implementation of the model for a minimal theory of biology, a discussion organizational closure and proper display of the results of the systemo, and results found through multiple runs of my system, with comparisons to those found by Fontana and Buss.

3.1 Minimal Theory of Biology Based on Multi-variable Lambda Calculus

The model for the minimal theory of biology is based on the lambda calculus, a theory of computation equally powerful to Turing machines. In the model, the elements are defined as λ-terms, a string defining both the name and the function of the element. The following is taken directly from (Fontana and Buss 1994). The symbols allowed in a λ-term are elements from an infinite supply of variables $V = \{x_1, x_2, ...\}$ the abstraction

symbol λ, and structural symbols "." and "(" ")". The set of λ-terms Λ is defined:

$$x \in V \Rightarrow x \in \Lambda \tag{2}$$

$$x \in V, M \in \Lambda \Rightarrow \lambda x.M \in \Lambda \text{ (abstraction)} \tag{3}$$

$$M \in \Lambda, N \in \Lambda \Rightarrow (M)N \in \Lambda \text{ (application)} \tag{4}$$

A variable x is bound if it occurs within a subterm P in an abstraction $\lambda x.P$, otherwise it is free, denoted $x \in \phi(P)$. The syntax of a λ-term can be transformed through the following axioms.

Substitution

$$(\lambda x.x)Q \to Q \tag{5}$$

$$(\lambda x.E)Q \to E, \text{ if } x \notin \phi(E) \tag{6}$$

$$(\lambda x \lambda y.E)Q \to \lambda y.(\lambda x.E)Q, \text{ if } x \neq y \text{ and } (x \notin \phi(E) y \notin \phi(Q)) \tag{7}$$

$$(\lambda x.(E_1)E_2)Q \to ((\lambda x.E_1)Q)(\lambda x.E_2)Q \tag{8}$$

Renaming

$$\lambda x.E \to \lambda z.(\lambda x.E)z, z \notin \phi(E) \tag{9}$$

If the substitution axioms cannot be applied to an expression it is said to be in normal form. Reduction to normal form is the essence of computation in the lambda calculus. Not all terms have a normal form, and the problem of determining whether a normal form exists for any term is the lambda calculus equivalent of the halting problem. However, if a normal form exists for a term, it is unique. Fontana and Buss introduce a regular naming form for the variables such that λ binds variables x_i sequentially in $i = 1, 2, 3, ...$ from left to right. Regular naming in combination with a normal form creates equivalence classes in the form of equivalent strings. Since reduction may not complete, or grow too large, a pragmatic reduction is computed instead, returning either a normal form or abandoning the reduction, resulting in an elastic collision, if the number of reduction steps or the size of the function grows too large.

The flow reactor is initialized with 1000 random functions. The random functions are created by recursively and randomly creating variables, abstractions, and applications (with the unstated condition that it forms a valid λ-term) with probabilities p_1, p_2, p_3 respectively followed by reduction to normal form. The collision rule can also be represented as a function within the system $\phi \equiv \lambda x.\lambda y.(x)y$ so the collision between elements A and B is mediated by $((\phi)A)B$ reduced to normal form.

3.2 Implementation

I have re-implemented the system in Scheme, a Lisp-like programming language based on the lambda calculus. This is made more convenient by a couple of changes to the syntax. Since the variables are named in a regular fashion, we can infer their name from the number of lambdas that precede them, thus for abstraction I change the rule of inference to

$$M \in \Lambda \Rightarrow \lambda M \in \Lambda \tag{10}$$

I also change how variables are referenced. Fontana and Buss implemented their system in C which is ammenable to labeling the variables from left to right. As a lambda-like language, however, scheme is better suited to recursive naming, so all my variables are named with lexical scope, so that $V = \{(x_0), (x_1), (x_2), ...\}$ where x denotes a variable and the following number represents its binding in terms of the number of lambdas that nest it. This is also known as a de Brujin index for the variables, with the stylistic choice of wrapping each index number in a list with the symbol x. For example in $\lambda\lambda(x_1)$, the variable (x_1) is bound to the λ on the left while in $\lambda\lambda(x_0)$, the variable (x_0) is bound to the λ on the right. Because reduction to normal form is equivalent to the computation of the lambda-like language, instead of applying axioms of reduction I created an interpreter for the lambda calculus language, evaluating terms in applicative order. Note also that the period becomes redundant in this notation. To fit properly in Scheme's syntax, all λ-terms are wrapped in parentheses, and applications are wrapped in additional parentheses. For example, the ω function, which when applied to itself reduces to itself applied to itself infinitely, is represented as $\lambda x_1.(x_1)x_1$ in Fontana and Buss' notation, but will be represented as $(\lambda((x_0))x_0)$ in mine.

In an application $((M)N)$, the reduction strategy works in applicative order, evaluating each term M and N as much as possible before applying the substitution. With de Brujin indexes, the correctness of variable names in the string is automatically enforced through explicit renaming rules. In an application $((\lambda M)N)$ all free variables in M are decreased by 1, while all variables bound by the outer λ in M are replaced with N, whose free variables are increased matching the number of new λ abstractions they are nested under (De Brujin, 1972). The interpreter then either creates a string in normal form or abandons the computation if the number of nested interpretations or the size of the expression exceeds preset conditions, a pragmatic boundary condition that prevents infinite reductions from stalling the program or consuming all memory.

Because in the flow reactor free variables in λ-terms don't reduce and only serve to clutter the reactor with expressions that are eventually removed, they are not allowed in the initial creation of random functions. This means that while creating a function, before an abstraction is created there is a p_1+p_2 chance of creating an abstraction. In the examples that follow, the pragmatic boundary conditions require that the tree representation of a λ-term does not exceed a depth of 20 and has fewer than 1000 nodes. Similarly, pragmatic boundary on the reduction rule is that the reduction doesn't reach a level of 20 nested recursions.

3.3 Display and Detection of Organizational Closures

How should we display and detect organizational closures within the flow reactor? The first of Varela's requirements is that "the processes are related as a network, so that they recursively depend on each other in the generation and realization of the processes themselves" (Varela 1979). Let us first say that the processes are the collision between elements in the flow reactor. All collisions among a set of elements in the flow reactor A will be generated and realized if all elements in A can be produced through those collisions: $A \subseteq \mathbf{m}(A)$, where $\mathbf{m}$ is the map from a set of elements to the set of all possible elements produced by collisions between the elements, as described above. A naive detection of organizational closures could look at the map for each subset of unique elements in the flow reactor to determine whether they form an organizational closure, but this would be wasteful considering that all elements of the powerset 2^n for n unique elements would be tested.

Instead we can apply the map $\mathbf{m}$ to the set of all unique elements in the flow reactor once, and create a directed bipartite network, which will be referred to as a *reaction graph*, with three edge types. One class of nodes represents the elements of the flow reactor. The other represents collisions between the elements. The two elements involved in a collision are connected by edges from the elements to the collision. Different edge types represent whether the element is the *operator* M or *operand* N in an application collision $((M)N)$. An edge from a collision to an element denotes that the element is the *product* of the collision. The problem of detecting an organizational closure then becomes the well known problem of detecting strongly connected components on a directed graph with an additional requirement. Tarjan's solution to this problem has a fast linear time-complexity of $O(|V| + |E|)$ where V is the set of vertices (nodes) on the graph and E is the set of edges. The additional requirement we must add is that for each collision within a strongly connected component, both of the elements involved in the collision are also included in the component. If the strongly connected component is still stronly connected once collisions not meeting this requirement are removed, the component is an organizational closure. In all my random simulations, after one million collisions there is either one or no organizational closures in the flow reactor.

There is a second requirement that Varela lists for an organization to be closed: "(the processes) constitute the system as a unity recognizable in the space (domain) in which the processes exist" (Varela 1979). This is a subtler point and would seem to justify the search for the strongly connected components of the reaction graph. A strongly connected component of a graph is a maximal one. No further nodes or edges can be included without loss of the property of being strongly connected. However, subgraphs of the strongly connected component may also be strongly connected, and meet the further requirement of containing all elements of its collisions. While there are no real limits on what the observer can distinguish as a unity, the idea that a strongly connected component is maximal within the larger reaction graph fits nicely with the idea of a unity recognizable in the larger space. If we could distinguish cells within a body in this sense, then we may say that the cell is not detected as an organizational closure. If the cell is removed and survives, it would be detected as an organizational

closure, but it may require or develop additional processes in its new medium to make up for the loss of the rest of the body, and these processes would be included in the organization as well.

We can loosen our definition to allow for more organizations. For example, if we consider the operator in the application collision to be process, then we can loosen the requirement that the operand must also be present in the strongly connected component. The operand then represents an input to the system, like food. Similarly, we can loosen the requirement on the operator. However, since in the application the operand may also be an operator within the operator, or the operand may be ignored (not referenced in the operator), there is not a true reason to prefer one position as process over the other. In most of the examples below there is no difference between closures with the strongest requirements, and those with looser requirements.

The reaction graph also offers a convenient means to display the network of processes. If we place the collision nodes as invisible nodes between the operand and the collision's product, we can produce the sorts of graphs Fontana and Buss produced for their Level 0 organizations, described below, automatically for all nodes. I concentrate on circular layouts of the elements in order to maximize space to describe relationships, although this quickly becomes too cluttered for easy interpretation in larger graphs. Transformations of an operand into another element are represented as solid directed arrows, while operators acting on an operand are displayed as blue dotted arrows targeting the edge between operand and product, the arrow head drawn closer to product to distinguish cases when the roles of operand and product can be reversed. Fontana and Buss also describe some organizations as grammatical rules, although even these description are described as difficult in larger organizations. However, I have not managed to produce an automatic grammar detection here.

4 Results

Fontana and Buss find a series of self-maintaining organizations that they categorize according to a number of levels. The only change needed to produce the various organizations of the first two levels is to alter the collision rule to remove certain syntaxes from the system by producing an elastic collisions whenever they occur. The probabilities for the recursive creation of each type of random expression are $p_1 = .50$, $p_2 = .25$, $p_3 = .25$, unless noted otherwise. The plots of time series sample the frequency of elements in the reactor every 100 collisions. Elastic collisions count as collisions in time series in order to ensure that the simulation terminates even in inert, or fully elastic, reactors.

4.1 Level 0

Level 0 organizations are characterized by n-membered elementary hyper-cycles of self-reproducing functions according to Fontana and Buss. In such an organization object i mod n copies object $(i + 1)$ mod n. While those sorts of organizations may be possibly sustained if the flow reactor is initiallized with their elements, in many runs of the

Label	λ-term	Frequency
$A1$	$(\lambda(x_0))$	1000

Table 1: Level 0, Run 12, Flow Reactor after 1 Million steps

randomly initialized flow-reactor, I have only observed the collapse of the reactor to a single self-copying function $(\lambda(x_0))$. The evolution of the reactor over time is shown in Figure 1, while the reaction graph is displayed in Figure 2. The legend for the nodes of the reaction graph along with their frequency in the flow reactor are given in Table 1. These figures aren't very interesting, as they show only the identity function that the reactor has reduced to, but they provide an easily interpreted example of the forms that the figures of later examples will take.

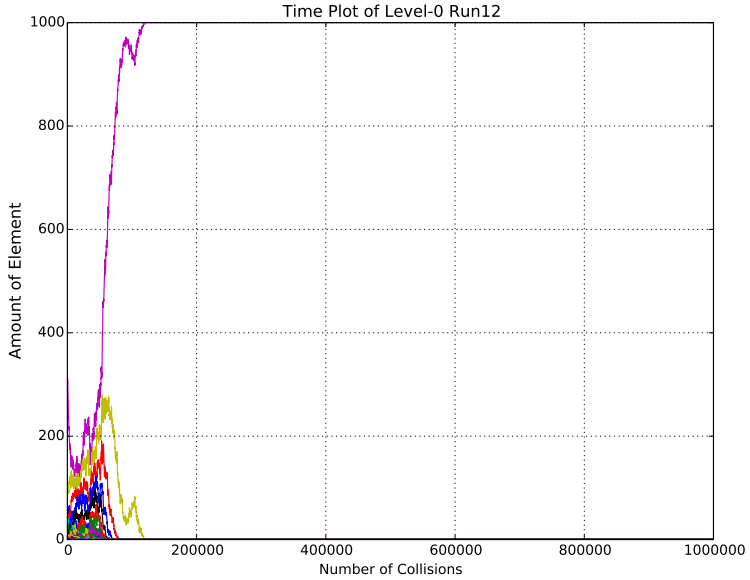

Figure 1: Level 0, Run 12, Element Frequency Over Number of Collisions

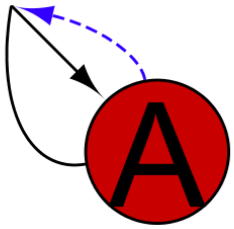

Figure 2: Level 0, Run 12, ReactionGraph

Label	λ-term	Frequency
$B1$	$(\lambda\lambda(x_0))$	974
$A1$	$(\lambda\lambda\lambda(x_0))$	26

Table 2: Level 1-1, Run 11, Flow Reactor after 1 Million steps

4.2 Level 1 - 1

Level 1 organizations are characterized by self-maintaining sets of objects. The most basic of these organizations, which I will refer to as level-1-1 organizations, are produced by labeling all collisions between objects A and B as elastic if they result in either of the original objects A or B. Since the identity function $(\lambda(x_0))$ can only reproduce an operand as an operator, and since the system tends to converge towards it, it is disallowed as a product, and any collision that produces it will hereafter be elastic. The functions in these organizations have the following form (Fontana & Buss 1994):

$$A_{i,j} \equiv \lambda x_1.\lambda x_2 \ldots \lambda x_i.x_j, j \leq i \tag{11}$$

In my syntax these appear as

$$A_{i,i-j-1} \equiv (\lambda\lambda ...i \text{ times} ... \lambda x_j), j < i \tag{12}$$

Since j refers to the preceding lambdas in an inverse manner, it will be the same for whole families of functions, thus I will refer a value of j as the base of a family. The smallest of such a family of λ-terms occurs when $j = i - 1$ (the -1 is due to a 0 starting index), which adds $i - 1$ λ abstractions to the operand it collides with. The collision of any other λ-term in the family with any element results in the removal of the outermost λ abstraction. This process is well described by Fontana and Buss's figure, so I include it here as Figure 3. The identity function, the smallest function for base $j = 0$ is disallowed, but if members of its family can come to dominate, the reactor will converge to an inert state composed only of the function $(\lambda\lambda(x_0))$. Even if the identity is not disallowed, then the smallest function of base $j = 0$ will have $i-1 = 0$ and additional lambdas cannot be added to move back up the organization. This occurs in Figures 4, 5 and Table 2.

More often a the base is some other number, frequently $j = 1$ or $j = 2$, and occasionally two bases will coexist after one million steps. In our above definition of organizational closure, when such a coexistence occurs, both families will be included in same closure. Typically there will only be one base. Such a typical run is displayed in Figures 6 and 7 and Table 3. Here the reaction graph is limited to the detected organizational closure. The missing node $M1$ is an element that can be produced by the others, but is not in fact in the reactor. In the reaction graph there are circular groupings of operator arrows around single elements. These correspond to the removal of a lambda from the operator, and are a visual indication that the idea of an operator as a process rather than the operand is not a particularly accurate one.

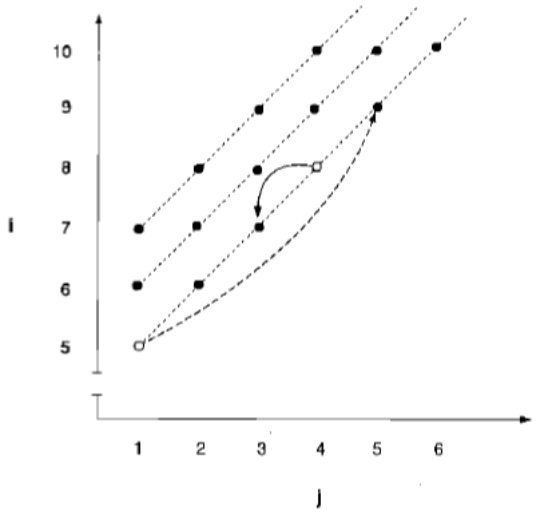

Figure 3: The simplest level 1 organization (Fontana & Buss 1994)

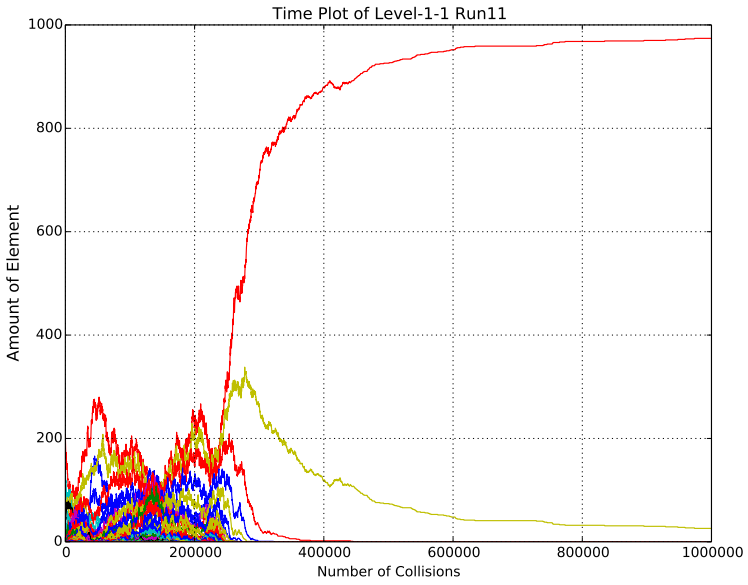

Figure 4: Level 1-1, Run 11, Element Frequency Over Number of Collisions

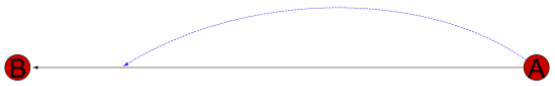

Figure 5: Level 1-1, Run 11, ReactionGraph

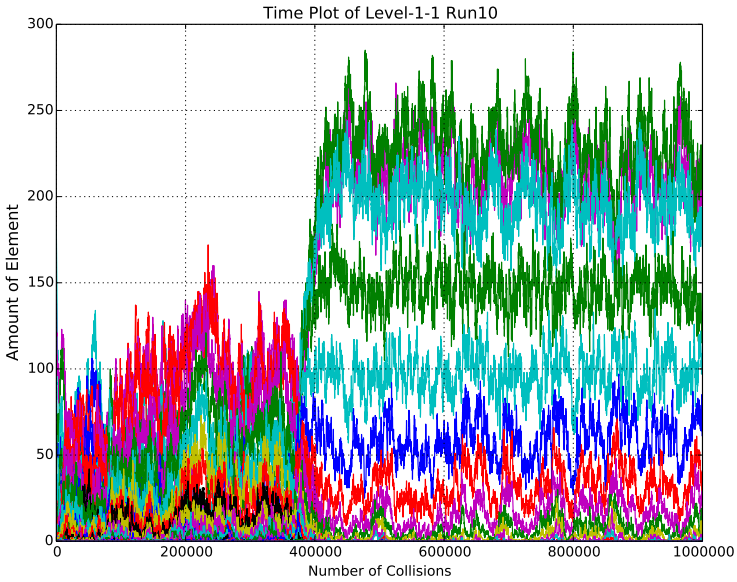

Figure 6: Level 1-1, Run 10, Element Frequency Over Number of Collisions

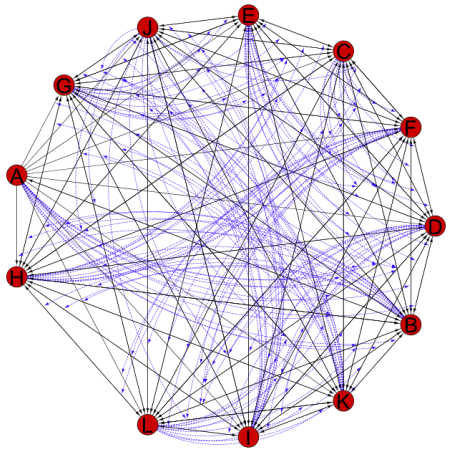

Figure 7: Level 1-1, Run 10, Closure Reaction Graph

Label	**λ-term**	**Frequency**
$I1$	$(\lambda\lambda\lambda(x_1))$	217
$J1$	$(\lambda\lambda(x_1))$	203
$D1$	$(\lambda\lambda\lambda\lambda(x_1))$	183
$E1$	$(\lambda\lambda\lambda\lambda\lambda(x_1))$	133
$G1$	$(\lambda\lambda\lambda\lambda\lambda\lambda(x_1))$	113
$K1$	$(\lambda\lambda\lambda\lambda\lambda\lambda\lambda(x_1))$	64
$L1$	$(\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda(x_1))$	36
$F1$	$(\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda(x_1))$	26
$C1$	$(\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda(x_1))$	13
$H1$	$(\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda(x_1))$	8
$B1$	$(\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda(x_1))$	3
$A1$	$(\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda(x_1))$	1
$M1$	$(\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda(x_1))$	0

Table 3: Level 1-1, Run 10, Flow Reactor after 1 Million steps

Label	λ-term	Frequency
$C1$	$(\lambda((x_0))\lambda(x_0))$	522
$A1$	$(\lambda((x_0))\lambda((x_0))\lambda(x_0))$	477
$B1$	$(\lambda((x_0))\lambda((x_0))\lambda((x_0))\lambda(x_0))$	1

Table 4: Level 1-2, Run 13, Flow Reactor after 1 Million steps

4.2.1 Level 1 - 2

Level-1-2 organizations arise if the conditions of level-1-1 are met and a pattern of three consecutive abstractions are disallowed from the system. Fontana and Buss describe an organizaation characterized by two families $\mathcal{A}$ and $\mathcal{B}$ of the forms (Fontana & Buss 1994):

$$A_i \equiv \lambda x_1.(x_1)\lambda x_2.(x_2)...\lambda x_{i+3}.\lambda x_{i+4}.(x_{i+4})\lambda x_{i+5}.(x_{i+5})x_{i+3}, i \in -2, -1, 0, 1, 2, ... \quad (13)$$

$$B_i \equiv \lambda x_1.(x_1)\lambda x_2.(x_2)...\lambda x_{i+1}.x_{i+1}, i \in 0, 1, 2, ... \quad (14)$$

In my notation, the corresponding families are written:

$$A_{i-2} \equiv (\lambda((x_0))\lambda((x_0))...i \text{ times}...\lambda\lambda((x_0))\lambda((x_0))(x_2)), i \in 0, 1, 2, ... \quad (15)$$

$$B_i \equiv (\lambda((x_0))\lambda((x_0))...i \text{ times}...\lambda(x0)), i \in 0, 1, 2, ... \quad (16)$$

The organization of these components is described through difference functions on their indices. An observer can specify grammatical laws equivalent to the operation of the lambda calculus on these objects, without reference to the operation of the lambda calculus. While my interpreter accurately produces the same results for collisions among these elements when specifying them, I have not observed this organization appearing from random initial conditions. Most often the reactor converges towards an inert state composed only of small elements of the B family. This is shown in Figures 8 and 9 and Table 4. This is one of few examples where loosening the requirement that the operands reside in the strongly connected component detects an organizational closure when the stronger requirement does not, this organizational closure is displayed in Figure 10. In other words, if the function $(\lambda((x_0))\lambda((x_0))\lambda((x_0))\lambda(x_0))$ is considered food rather than a part of the organization there is an organizational closure in this example.

The only example of stable, reactive organization I found is displayed in Figures 11 and 12 and Table 5. While there is a complex organizational closure involving the entire reaction graph, there are none of the A family functions Fontana and Buss discuss. This example has fewer of the circles of operator targets, common in level-1-1, where application to any function would result in the same output by removing a lambda in the operator. There are a number of functions that immediately apply their first argument, so that they all converge to the same product when applied to the same operand, as see by clusters of operator arrow heads on the same edge. However, there are a number of productions that escape a simple explanation, and require more thought to describe. Such as $C1$ applied to $G1$ producing $I1$, which results in an order of applications of $G1$ to itself wrapped in abstractions until eventual reducion to the simple expression of $I1$.

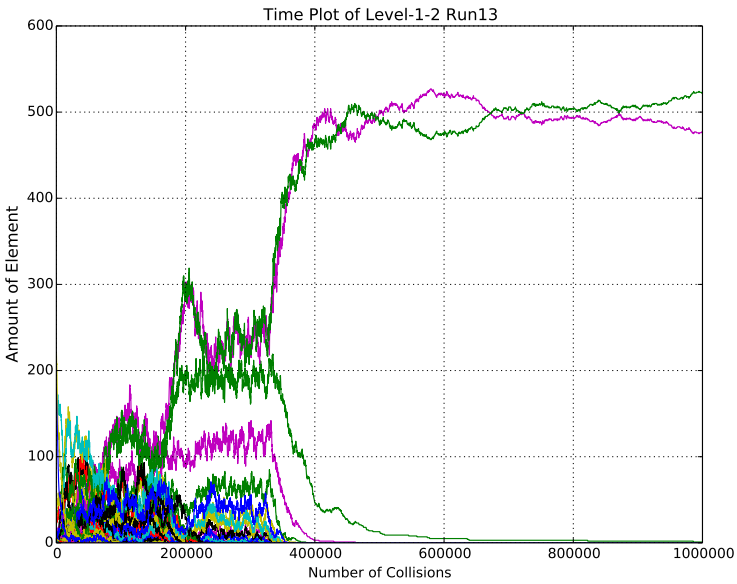

Figure 8: Level 1-2, Run 13, Element Frequency Over Number of Collisions

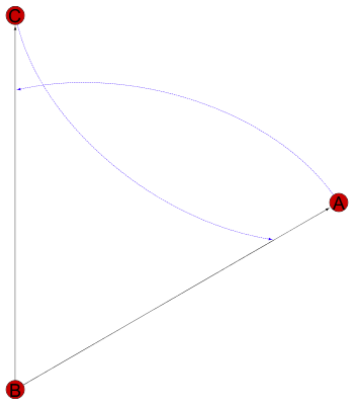

Figure 9: Level 1-2, Run 13, Reaction Graph

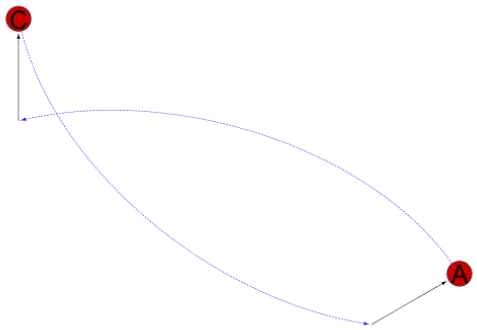

Figure 10: Level 1-2, Run 13, Operator Closure Reaction Graph

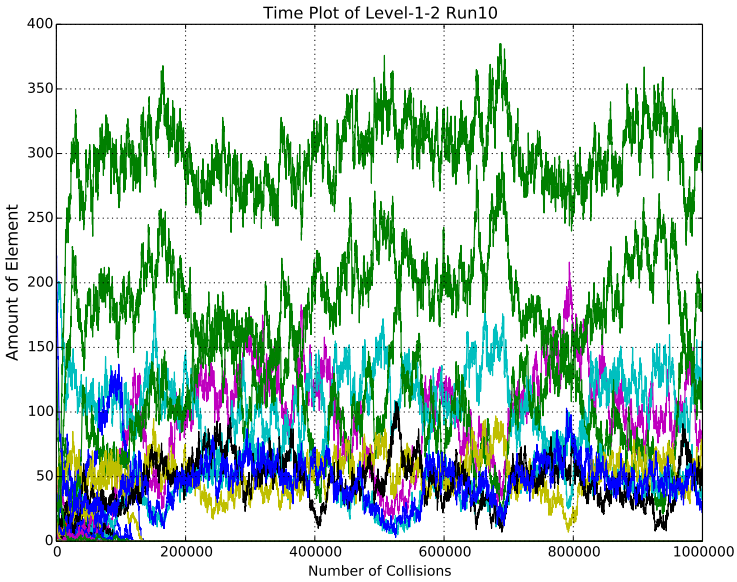

Figure 11: Level 1-2, Run 10, Element Frequency Over Number of Collisions

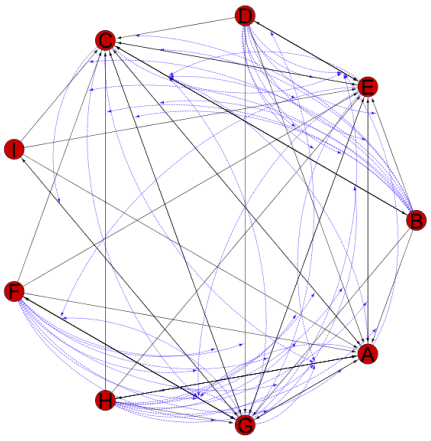

Figure 12: Level 1-2, Run 10, Reaction Graph

Label	**λ-term**	**Frequency**
$G1$	$(\lambda((x_0))\lambda\lambda(x_2))$	289
$F1$	$(\lambda\lambda((x_0))\lambda\lambda(x_2))$	189
$E1$	$(\lambda((x_0))(x_0))$	147
$A1$	$(\lambda((x_0))\lambda(x_1))$	115
$C1$	$(\lambda((x_0))(((x_0))\lambda(x_1))\lambda(x_0))$	70
$D1$	$(\lambda\lambda((x_0))(x_0))$	66
$H1$	$(\lambda\lambda((x_0))\lambda(x_1))$	56
$B1$	$(\lambda\lambda((x_0))(((x_0))\lambda(x_1))\lambda(x_0))$	40
$I1$	$(\lambda\lambda(x_0))$	28

Table 5: Level 1-2, Run 10, Flow Reactor after 1 Million steps

4.2.2 Level 1 - 3

Level-1-3 organizations arise when the conditions for level-1-2 are applied and the pattern $\lambda x_i.(x_j)\lambda x_k.(x_l)$ in Fontana and Buss' notation or $(\lambda((x_j))\lambda((x_l)))$ in mine is barred from appearing in objects. This condition prevents the inert reactors described in level-1-2. This level produces many varied organizations, ranging from small to large numbers of λ-terms, typically quite complex. I ran more simulations with these conditions than any other level, and included a variety of different probabilities for generating random functions: Uniform=$(p_1 = .33, p_2 = .33, p_3 = .34)$, HeavyApp=$(p_1 = .25, p_2 = .25, p_3 = .5)$, and LowVar=$(p_1 = .2, p_2 = .4, p_3 = .4)$. One of the simplest organizations in terms of the number of unique λ-terms is shown in Figures 13 and 14 and Table 6. Although the productions of A1 and E1 can only be reached through an operator edge, they still fit within the connected component. However, if the physics of the flow reactor were changed such that the operand was always selected as the element to be replaced after a collision, the store of A1 and E1 would eventually be removed through dilution. Another organization found through a different set of intial conditions and random function generation probabilities is shown in Figures 15 and 16 and Table 7.

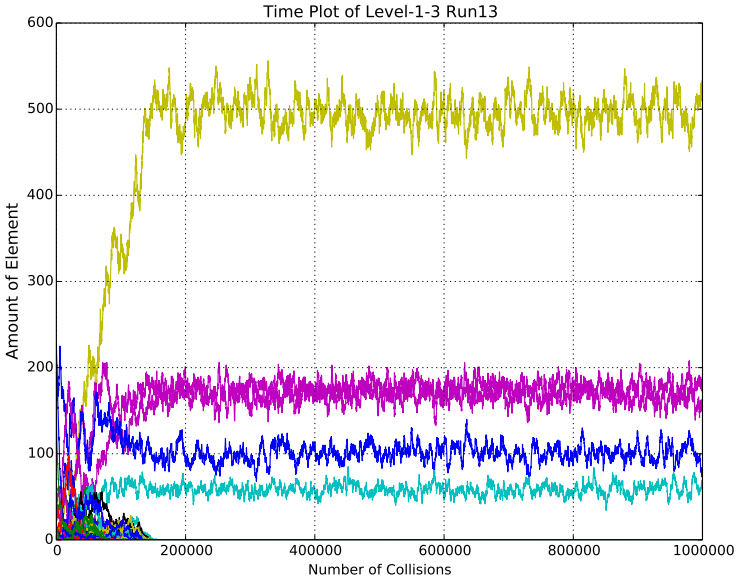

Figure 13: Level 1-3, Run 13, Element Frequency Over Number of Collisions

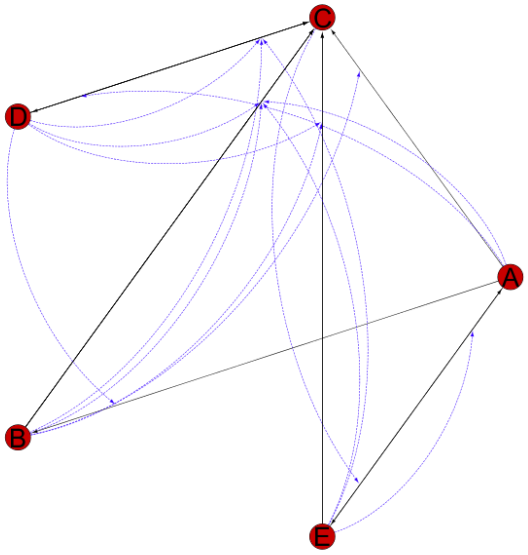

Figure 14: Level 1-3, Run 13, Reaction Graph

Label	**λ-term**	**Frequency**
$C1$	$(\lambda(((x_0))\lambda\lambda((x_1))((x_0))(x_1))((x_0))((x_0))\lambda\lambda((x_1))((x_0))(x_1))$	526
$D1$	$(\lambda(((x_0))\lambda((x_0))\lambda\lambda((x_1))((x_0))(x_1))\lambda\lambda((x_1))((x_0))(x_1))$	173
$E1$	$(\lambda((x_0))((x_0))\lambda\lambda((x_0))(x_1))$	157
$A1$	$(\lambda((x_0))\lambda\lambda((x_0))(x_1))$	79
$B1$	$(\lambda\lambda(((x_0))\lambda\lambda((x_1))((x_0))(x_1))((x_0))((x_0))\lambda\lambda((x_1))((x_0))(x_1))$	65

Table 6: Level 1-3, Run 13, Flow Reactor after 1 Million steps

Label	**λ-term**	**Frequency**
$E1$	$(\lambda((((x_0))\lambda((((x_0))\lambda\lambda(x_1))(x_0))(x_0))(x_0))(x_0))$	196
$A1$	$(\lambda((((x_0))\lambda\lambda(x_1))(x_0))(x_0))$	186
$F1$	$(\lambda\lambda((((x_0))\lambda((((x_0))\lambda\lambda(x_1))(x_0))(x_0))(x_0))(x_0))$	138
$B1$	$(\lambda\lambda((((x_0))\lambda\lambda(x_1))(x_0))(x_0))$	136
$G1$	$(\lambda((x_0))\lambda\lambda(x_1))$	112
$D1$	$(\lambda((x_0))(x_0))$	109
$H1$	$(\lambda\lambda((x_0))\lambda\lambda(x_1))$	62
$C1$	$(\lambda\lambda((x_0))(x_0))$	61

Table 7: Level 1-3, Run 16-HeavyApp, Flow Reactor after 1 Million steps

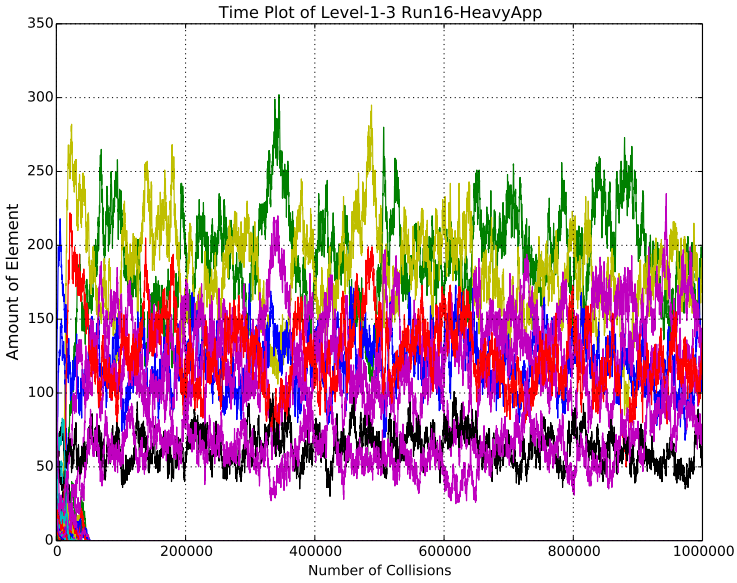

Figure 15: Level 1-3, Run 16-HeavyApp, Element Frequency Over Number of Collisions

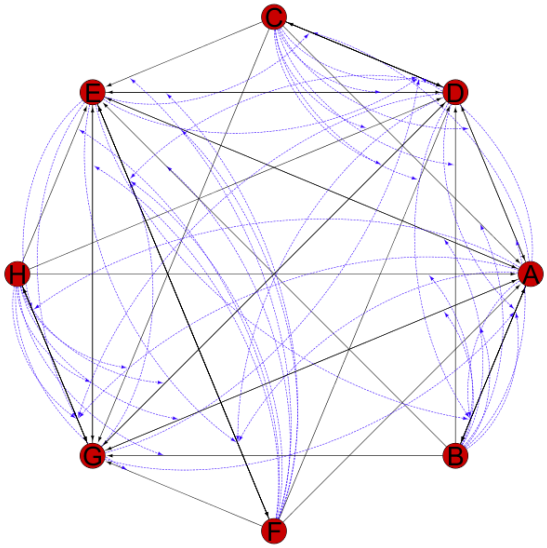

Figure 16: Level 1-3, Run 16-HeavyApp, Reaction Graph

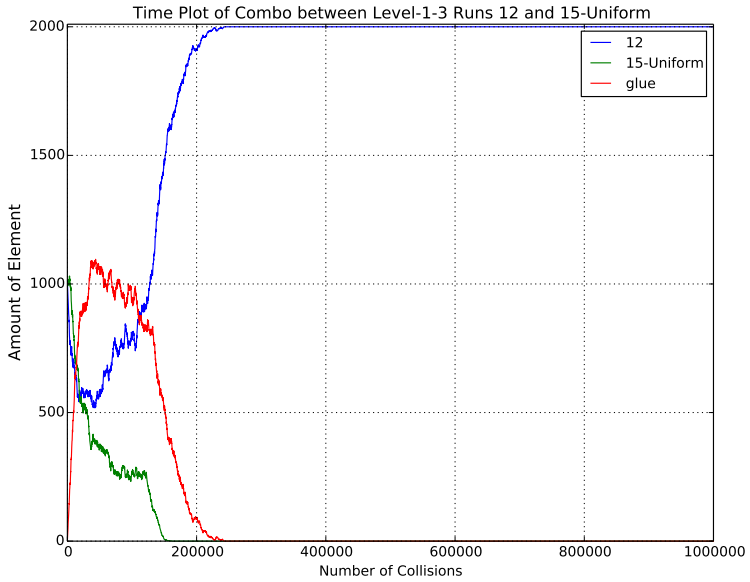

Figure 17: Level 2, Combination of Runs 12 and 15-Uniform, Reaction Graph

4.2.3 Level 2

Level 2 organizations arise as meta-organizations of level 1 organizations. If two level 1 organizations are placed in the same reactor there are three possibilities. One comes to dominate and the other is flushed out of the reactor by dilution, there is coexistence (until one is randomly flushed) without stable interaction, or a meta-organization arises. These meta-organizations are stabilized by a "glue" produced by objects of each organization acting upon the objects of the other organization. In order for both organizations and the glue to be stabilized, there must be self-maintaining sets between them, i.e. organizational closure, in addition to the organizational closure of the individual level 1 organizations. Fontana and Buss found that level 2 organizations are easier to achieve if different collision rules are specified for the glue and the level 1 organizations. However, the distinction of separate collision rules in my implementation is not simple and has not been finished. Such separate collision rules also do not have a clear justification, although it could perhaps represent the constraint of a larger body acting on elements in a medium outside of the two level 1 organizations. Perhaps because of this, I was unable to produce a clear Level 2 organization unless there were a large number of elements shared between the two level 1 organizations, as shown in Figure 18. Otherwise one organization comes to dominate, as shown in Figure 17.

If the level 1 organizations share a common element, I include its count in both. This natural integration of organizationally closed systems into a hierarchy is a fascinating result of this model I regret not observing myself. The meta-organization is just as

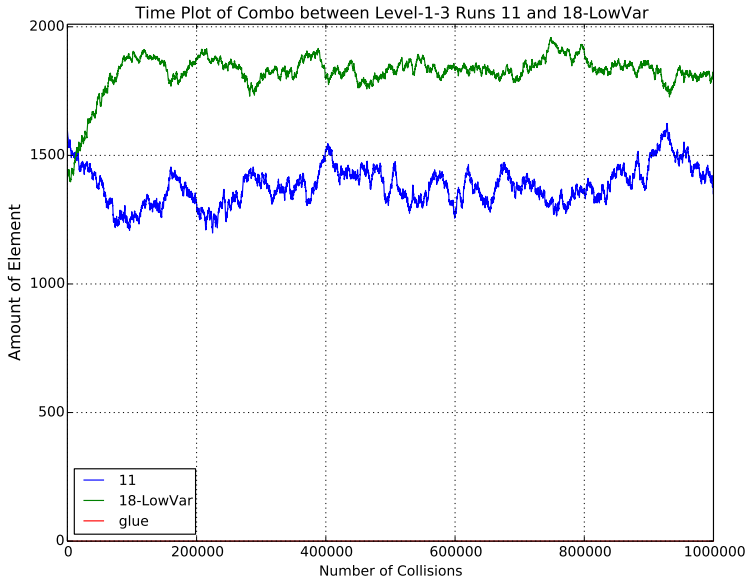

Figure 18: Level 2, Combination of Runs 12 and 18-LowVar, Reaction Graph

organizationally closed as its component organizations while they remain nonetheless strongly connected, and thus possibly autonomous. If definition of organizational closure as the strongly connected components of the larger space is accepted however, we could not say these components are autonomous themselves without observing them as in a separate medium. That organizations could theoretically lose the ability to produce some of their own components shows how their autonomy can be subjugated to the larger system, or in other words, defined in relation to a medium that is the higher level organization. The glue is also a nice example of structural coupling that exists so long as autonoumous organizations have recurrent interactions, while not itself being autonomous on its own.

4.3 Internal Collision Rules

As mentioned before, the application collision rule can be represented in the system itself, as the λ-term, rewritten in my notation: $\phi \equiv (\lambda\lambda((x_1))(x_0))$, and its application is the reduction of the term $(((\phi)A)B)$ on two separate elements A and B. In fact, this can be interpreted as two collisions in sequence between $(((\phi)\phi)A) \rightarrow C$ and $(((\phi)C)B)$. This suggests a different way to separate process and component. We can allow the collision rule to be selected from the flow reactor at random, then apply it immediately to a number of randomly selected terms equal to the number leading abstractions in the collision rule. This does not allow for any terms that were not allowed before, but increases the probability of certain reactions occuring. For example, for a collision rule

λ-term	Frequency
$(\lambda\lambda\lambda(x_1))$	1000

Table 8: Internal Collision Rules Level 0, Run 10, Flow Reactor after 1 Million steps

with two leading abstractions and n_0 duplicates of itself in the reactor, the chance of colliding two terms in sequence, with n_1 and n_2 duplicates respectively, is $\frac{n_0}{N} \times \frac{n_1}{N-1} \times (\frac{n_3+1}{N} \times (\frac{N-1}{N} \times \frac{n_2}{N-1} + \frac{1}{N} \times \frac{n_2-1}{N-1}))$, where n_3 is the number of duplicates of the product of n_0 and n_1 already in the reactor. The expression is the probability that the collision rule is selected, and the first term is selected, and then the product is selected, and then the second term is selected, whether or not one of its duplicates was removed by dilution. Whereas if all leading abstractions are satisfied at once. We have a probability of $\frac{n_0}{N} \times \frac{n_1}{N-1} \times \frac{n_2}{N-2}$.

This version of λ-term as collision rule uses a meta-collision rule much harder to create in the simple λ calculus of the model, since it involves counting lambdas. However, an element acting as a collision rule acts more as a process or catalyst than a simple operator could; no application results only in the simple removal of a leading λ. A number of initial simulations with such *internal collision* rules and the different boundary conditions explained in the previous sections have been completed. However, new tools have yet to be developed to create a comprehensible reaction graph and define organizational closures under this system. The timeseries of runs with internal collisions are shown in Figures 19, 20, 21, and 22, while the *lambda*-terms and frequencies after one million steps are shown in Tables 8, 9, 10, and 11.

These intial results suggest that using internal terms as collision rules seems to maintain greater variety and more complicated expressions. For example, the level-0 organization using internal collisions still stabilizes as a single self-replicating λ-term. However, the term is $(\lambda\lambda\lambda(x_1))$ rather than the usual identity $(\lambda(x_0))$. It is easy to understand why, (x_1) refers to the second element in the collision, while the first and third elements remove an abstraction. When placed in a population composed only of its duplicate, this element thus replicates itself (as only itself can be its second argument). In fact, any of the typical elements of level-1-1 organizations described by Fontana and Buss would be an identity function for one of its arguments. Since duplications are not allowed on level-1-1, the original level-1-1 families should thus never dominate a reactor using internal collisions.

λ-term	Frequency
$(\lambda\lambda((x_0))\lambda\lambda((x_0))(x_1))$	108
$(\lambda((x_0))\lambda\lambda((x_0))(x_1))$	100
$(\lambda\lambda\lambda((x_2))\lambda\lambda((x_0))(x_1))$	98
$(\lambda\lambda\lambda((x_0))\lambda\lambda((x_0))(x_1))$	89
$(\lambda\lambda\lambda\lambda((x_2))\lambda\lambda((x_0))(x_1))$	80
$(\lambda\lambda\lambda((x_2))\lambda\lambda\lambda\lambda\lambda((x_2))(x_4))$	54
$(\lambda\lambda\lambda\lambda\lambda((x_2))\lambda\lambda((x_0))(x_1))$	53
$(\lambda\lambda((x_0))\lambda\lambda\lambda\lambda\lambda((x_2))(x_4))$	48
$(\lambda((x_0))\lambda\lambda\lambda\lambda\lambda((x_2))(x_4))$	44
$(\lambda\lambda\lambda\lambda\lambda\lambda((x_2))\lambda\lambda((x_0))(x_1))$	44
$(\lambda\lambda\lambda((x_0))\lambda\lambda\lambda\lambda\lambda((x_2))(x_4))$	40
$(\lambda\lambda\lambda((x_2))\lambda((x_0))\lambda\lambda((x_0))(x_1))$	36
$(\lambda((x_0))\lambda((x_0))\lambda\lambda((x_0))(x_1))$	31
$(\lambda\lambda\lambda\lambda((x_2))\lambda((x_0))\lambda\lambda((x_0))(x_1))$	30
$(\lambda\lambda((x_0))\lambda((x_0))\lambda\lambda((x_0))(x_1))$	30
$(\lambda\lambda\lambda\lambda((x_2))\lambda\lambda\lambda\lambda\lambda((x_2))(x_4))$	26
$(\lambda\lambda\lambda((x_0))\lambda((x_0))\lambda\lambda((x_0))(x_1))$	26
$(\lambda\lambda\lambda\lambda\lambda\lambda((x_2))\lambda\lambda\lambda\lambda\lambda((x_2))(x_4))$	18
$(\lambda\lambda\lambda\lambda\lambda((x_2))\lambda((x_0))\lambda\lambda((x_0))(x_1))$	17
$(\lambda\lambda\lambda\lambda\lambda((x_2))\lambda\lambda\lambda\lambda\lambda((x_2))(x_4))$	16
$(\lambda\lambda\lambda\lambda\lambda\lambda((x_2))\lambda((x_0))\lambda\lambda((x_0))(x_1))$	12

Table 9: Internal Collision Rules Level 1-1, Run 10, Flow Reactor after 1 Million steps

λ-term	Frequency
$(\lambda((x_0))\lambda\lambda((x_0))(x_1))$	530
$(\lambda(((x_0))\lambda\lambda((x_1))((x_0))(x_1))((x_0))((x_0))\lambda\lambda((x_1))((x_0))(x_1))$	444
$(\lambda((x_0))((x_0))\lambda\lambda((x_0))(x_1))$	418
$(\lambda(((x_0))\lambda((x_0))\lambda\lambda((x_1))((x_0))(x_1))\lambda\lambda((x_1))((x_0))(x_1))$	231
$(\lambda((x_0))\lambda((x_0))\lambda\lambda((x_0))(x_1))$	210
$(\lambda\lambda(((x_0))\lambda\lambda((x_1))((x_0))(x_1))((x_0))((x_0))\lambda\lambda((x_1))((x_0))(x_1))$	167

Table 10: Internal Collision Rules Level 1-2, Run 10, Flow Reactor after 1 Million steps

λ-term	Frequency
$(\lambda((((x_0))\lambda\lambda((x_0))(x_0))\lambda\lambda((x_0))(x_0))\lambda\lambda((((x_0))\lambda\lambda((x_0))(x_0))\lambda\lambda((x_0))(x_0))(x_1))$	470
$(\lambda(((x_0))\lambda((x_0))\lambda(x_2))\lambda(x_1))$	319
$(\lambda((x_0))\lambda\lambda(((x_0))\lambda((x_0))\lambda(x_2))\lambda(x_1))$	251
$(\lambda((x_0))(x_0))$	243
$(\lambda\lambda(x_1))$	122
$(\lambda\lambda((x_0))\lambda\lambda(((x_0))\lambda((x_0))\lambda(x_2))\lambda(x_1))$	110
$(\lambda((x_0))\lambda\lambda(((x_0))\lambda\lambda(x_1))(x_2))$	74
$(\lambda(((x_0))\lambda\lambda(x_1))\lambda((x_0))\lambda\lambda(((x_0))\lambda\lambda(x_1))(x_2))$	60
$(\lambda\lambda(((x_0))\lambda((x_0))\lambda(x_2))\lambda(x_1))$	59
$(\lambda(((x_0))\lambda\lambda(x_1))\lambda((x_0))\lambda\lambda((x_0))\lambda\lambda(((x_0))\lambda\lambda(x_1))(x_2))$	54
$(\lambda\lambda((x_0))\lambda\lambda(((x_0))\lambda\lambda(x_1))(x_2))$	51
$(\lambda(((x_0))\lambda\lambda(x_1))\lambda(((x_0))\lambda((x_0))\lambda(x_2))\lambda(x_1))$	38
$(\lambda((x_0))\lambda\lambda((x_0))\lambda\lambda(((x_0))\lambda\lambda(x_1))(x_2))$	32
$(\lambda(((x_0))\lambda\lambda(x_1))\lambda((x_0))(x_0))$	28
$(\lambda\lambda((x_0))\lambda\lambda((x_0))\lambda\lambda(((x_0))\lambda\lambda(x_1))(x_2))$	24
$(\lambda\lambda(((x_0))\lambda\lambda(x_1))\lambda\lambda(x_1))$	20
$(\lambda(((x_0))\lambda\lambda(x_1))\lambda((x_0))\lambda\lambda(((x_0))\lambda((x_0))\lambda(x_2))\lambda(x_1))$	19
$(\lambda(((x_0))\lambda\lambda(x_1))\lambda\lambda(x_1))$	15
$(\lambda\lambda((x_0))(x_0))$	11

Table 11: Internal Collision Rules Level 1-3, Run 10, Flow Reactor after 1 Million steps

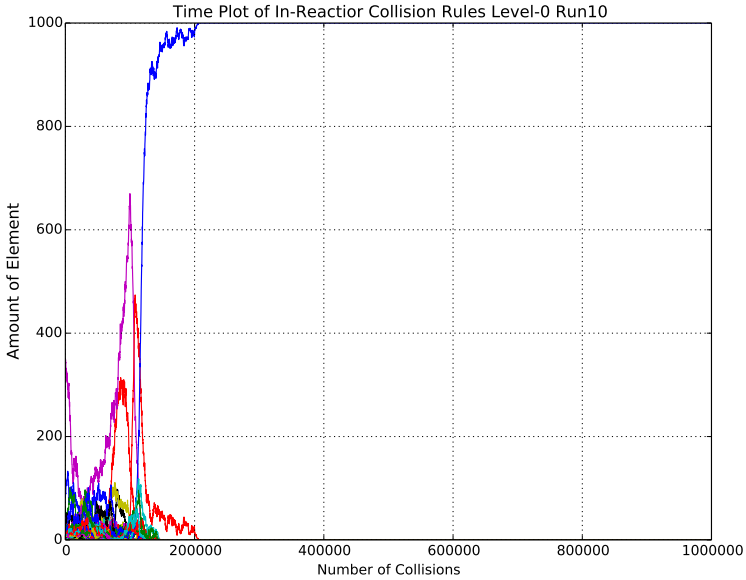

Figure 19: Internal Collision Rules Level 0, Run 10, Element Frequency Over Number of Collisions

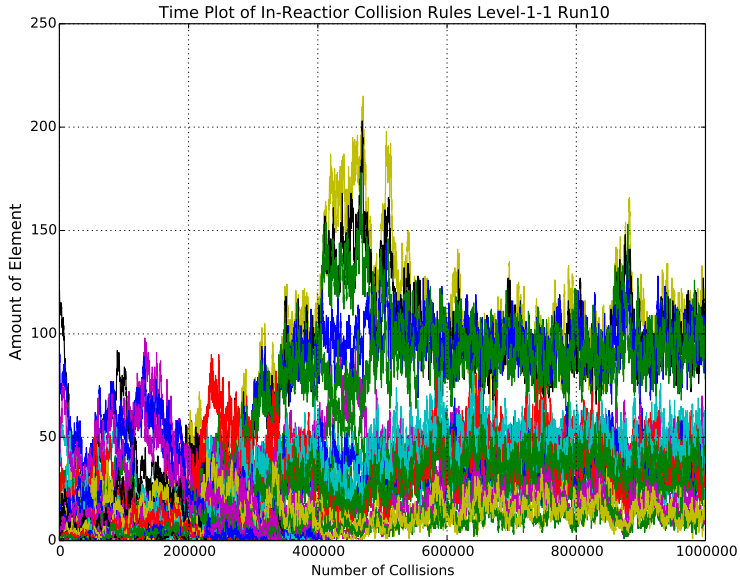

Figure 20: Internal Collision Rules Level 1-1, Run 10, Element Frequency Over Number of Collisions

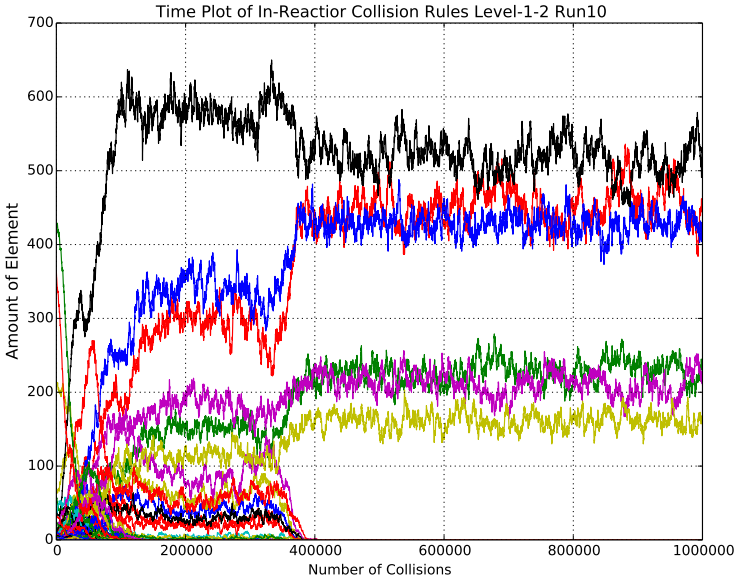

Figure 21: Internal Collision Rules Level 1-2, Run 10, Element Frequency Over Number of Collisions

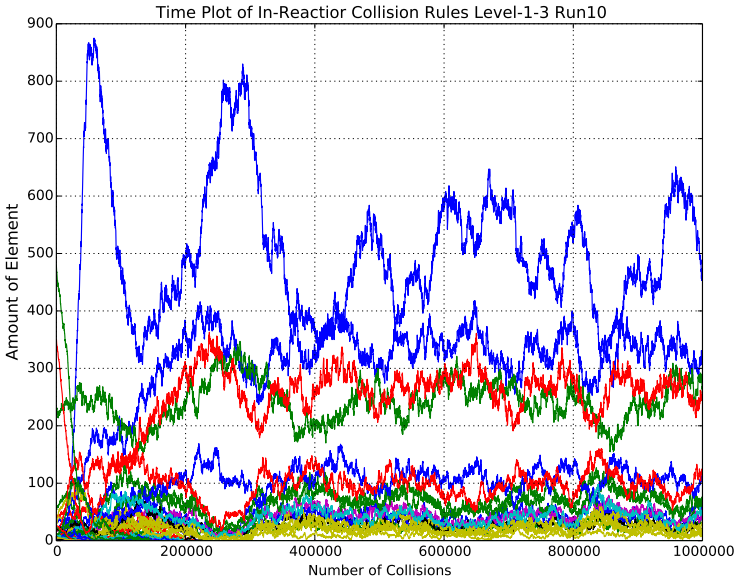

Figure 22: Internal Collision Rules Level 1-3, Run 10, Element Frequency Over Number of Collisions

5 Discussion

The minimal model for a theory of biology can clarify many aspects of organizational closure. With a few changes to definitions, not all components need to be produced by the system in order for closure. This requires a loosening of the strong requirement that strongly connected components in the reaction graph contain both operator and operand of all collisions. Waste products are naturally culled from this closure. However, we have seen that an element acting as operator is not an adequate interpretation of process in this system. A better interpretation may be as collision rule, but the proper definition of organizational closure in such a system still needs to be defined. With the simpler collision rule and definition of organizational closure as the strongly connected components with the reaction graph of a flow reactor state we were able to automatically detect a variety of organizations from many randomized simulations. These organizations can be displayed as bipartite networks with three edge types, and shown over time in line plots.

Components can realize processes when their structure is interpreted as determining an interaction. This interaction can be specified in the meta-domain of an observer who distinguishes the interactions of the components by grammatical rules, in this way avoiding reference to the deterministic laws that realize the processes. However, some aspects of the algorithm running the system can still seep in unexpectedly: boundary conditions and a reactor system determining which interactions occur between components.

Implementing the entire system in a lambda-like language raises the possibility for some interesting extensions. In principle, the interpreter and the reactor system can be reduced to λ-terms of the form that objects within the system take. The use of continuations and tail-recursion also mean that while the structure of a function may be a tree, control of its execution can be determined by the functions themselves. One can imagine a system in which its own interpretation and physics are also explicitly components of the system, and executed according to control specified by its own execution. Such a system would be difficult to implement without endless loops, but an expansion on the system with internal collision rules might be a good first step.

6 Works Cited

Beer, R.D. (in press). Characterizing autopoiesis in the Game of Life. To appear in Artificial Life

Beer, R.D. (2014). The cognitive domain of a glider in the Game of Life. Artificial Life 20(2):183-206.

De Bruijn, N. G. (1972). Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. In Indagationes Mathematicae (Proceedings) 75(5):381-392. North-Holland.

Fontana, W. (1991). Algorithmic chemistry. Ed. C G Langton et al. Artificial life II 11:159209.

Fontana W. & Buss, L. (1994). 'The arrival of the fittest: Toward a theory of biological organization. Bulletin of Mathematical Biology 56(1): 1-64.

Maturana, H.R. (1980). Excerpt from Autopoiesis and Cognition (Introduction). Dordrecht: D. Reidel.

Maturana, H.R. & Varela, F.J. (1987). Excerpts from The Tree of Knowledge (pp. 74-75, 95-103). Shambhala.

Maturana, H.R. (1988). Ontology of observing (Sections 1-6). Conference Workbook, American Society for Cybernetics Conference, Felton, CA.

Tarjan, R. (1972). Depth-first search and linear graph algorithms. SIAM journal on computing, 1(2), 146-160.

Thompson, E. (2007). Excerpt from Mind in Life (Chapter 5). MIT Press.

Varela, F.J. (1979). Excerpts from Principles of Biological Autonomy (Chapters 2, 3, 4, 6, and 7). North Holland.

Varela, F.J. (1997). Patterns of Life: Intertwining identity and cognition. Brain and Cognition 34:72-87.

Weber, A. & Varela, F.J. (2002). Life after Kant: Natural purposes and the autopoietic foundation of biological individuality. Phenomenology and the Cognitive Sciences 1(2): 97-125.