Rosen's *(M,R)* System as an X-Machine

Michael L. Palmer[1], Richard A. Williams[2] and Derek Gatherer[1]*

[1]Division of Biomedical & Life Sciences, Faculty of Health & Medicine, Lancaster University, Lancaster LA1 4YW, UK

[2]Department of Management Science, Management School, Lancaster University, Lancaster LA1 4YW, UK

*Corresponding author

Email: d.gatherer@lancaster.ac.uk

Abstract

Robert Rosen's *(M,R)* system is an abstract biological network architecture that is allegedly both irreducible to sub-models of its component states and non-computable on a Turing machine. *(M,R)* stands as an obstacle to both reductionist and mechanistic presentations of systems biology, principally due to its self-referential structure. If *(M,R)* has the properties claimed for it, computational systems biology will not be possible, or at best will be a science of approximate simulations rather than accurate models. Several attempts have been made, at both empirical and theoretical levels, to disprove this assertion by instantiating *(M,R)* in software architectures. So far, these efforts have been inconclusive. In this paper, we attempt to demonstrate why - by showing how both finite state machine and stream X-machine formal architectures fail to capture the self-referential requirements of *(M,R)*. We then show that a solution may be found in communicating X-machines, which remove self-reference using parallel computation, and then synthesize such machine architectures with object-orientation to create a formal basis for future software instantiations of *(M,R)* systems.

1. Introduction

The quest for mechanistic explanation in biology reflects a long-standing commitment to avoid the error of Molière’s physician, who explained opium’s sleepinducing properties as being caused by its *virtus dormitiva* (Molière, 1673). Mechanism asks the question: "*how* does it work?" and expects a non-tautologous answer couched in some kind of machine-like analogy. If the mechanistic explanation is also a reductionist one, it will situate that machine-like analogy at a lower level of biological organization. "How does an organism work?" might be explained in terms of the mechanism of organs; "how does an organ work?" in terms of the mechanism of cells; and "how do cells work?" in terms of molecular mechanisms. Intermediate levels are easy to insert – gene or metabolic regulatory networks might be placed between molecules and cells, or organelles between cells and molecules. The layered hierarchy of explanations is mirrored by a corresponding hierarchy of research disciplines, from population biologists at the top, through organismal zoologists and botanists to physiologists, then cell biologists, systems biologists and biochemists, with molecular biophysicists occupying the layer where biology shades imperceptibly into quantum organic chemistry.

The concept of levels of understanding of the natural world and their corresponding inter-dependent allocation of scientific labour goes as far back as Auguste Comte in the early 19th century (Comte, 1830; Lenzer, 1998), and a recognisably modern formulation emerged from the interwar Vienna Circle group of philosophers (Carnap, 1934), but its central place in the minds of modern biologists was finally cemented by Francis Crick (1966; 1981) and Jacques Monod (1971). Such reductionism has always had its critics (Elsasser, 1998; Polanyi, 1968; Rosen, 1991; Waddington, 1968), and their successors have grown bolder since the advent of an explicitly anti-reductionist strand in systems biology (reviews by Gatherer, 2010; Mazzocchi, 2012).

Even if current "how does it work?" questions in systems biology can no longer rely so heavily on reductionist answers, it is harder to dispense with mechanistic ones. Even if a modern systems biologist does not believe that the function of a particular regulatory network can be understood in terms of a composite understanding of its parts, nevertheless a non-reductive explanation will still be likely to contain machine-like analogies of some kind. The roots of mechanistic explanation in biology are even deeper than those of reductionism, perhaps as far back as the 17^{th} century (reviewed by Letelier et al., 2011) – otherwise the audiences of 1673 could scarcely have appreciated Molière's joke concerning *virtus dormitiva* - and were completely in the ascendency by the early 20^{th} century (Loeb, 1912). In the era of molecular biology, opposition to mechanism has been sporadic and muted.

Robert Rosen made it his life's work to question both reductionist and mechanist strategies in biology. Developing the mathematical techniques of relational biology originated by Rashevsky (1973), Rosen conceived an abstract model, *(M,R)*, always written with brackets and usually in italics (Figure 1), which he claimed encapsulated the properties of a living system but was irreducible to its component parts (Rosen, 1964a; 1964b; 1966; 1991; 2000). Goudsmit (2007) redrew the *(M,R)* diagram in a way that is more comprehensible to biochemists, implicitly recasting *(M,R)* as a representation of a biochemical network consisting of three reactions, each of which produces a catalyst for one of the other reactions. Rosen's intentions were more general, presenting *(M,R)* as consisting of three broad processes found in all living systems: metabolism, repair and replication. Metabolism is represented by the $A \rightarrow B$

process, repair by B→*f* and replication by *f*→φ, generating respectively the catalysts necessary for metabolism, and in turn the catalysts for synthesis of those catalysts.

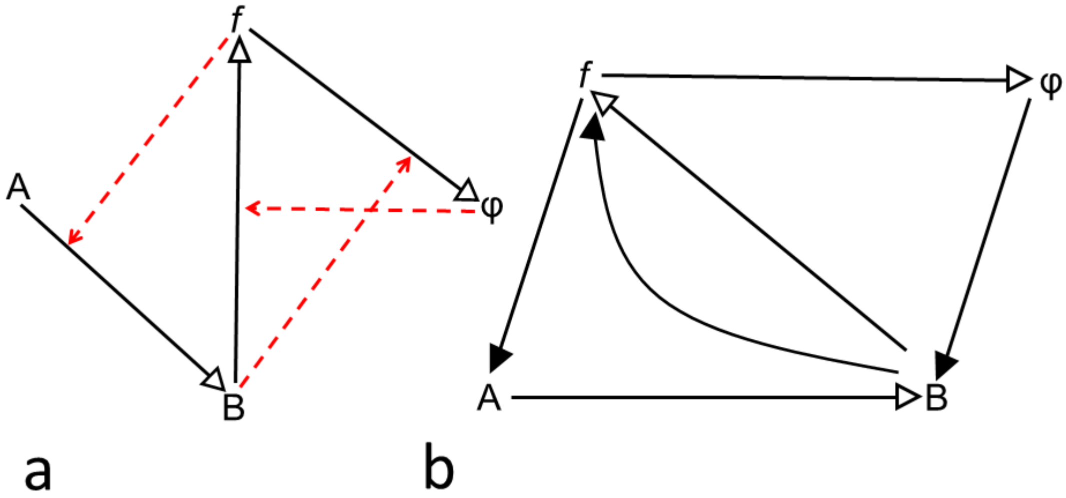

Figure 1 a: The Goudsmit representation of the *(M,R)* system. b: *(M,R)* diagram of Rosen. In the Goudsmit representation, productive reactions are shown using the black arrows and catalytic requirements using the red dotted arrows. In the *(M,R)* diagram of Rosen, the productive reactions are presented as open-headed arrows and the catalytic reactions as fill-headed arrows. The placement of the catalytic arrowheads is also on the substrate of the productive reaction.

The essence of Rosen's argument (Rosen, 1991) is that although each of the components of *(M,R)* can be understood as a machine, and therefore may be susceptible to mechanistic explanation, the whole cannot and may not. Furthermore, a model of the whole cannot be built additively from models of the components. *(M,R)* is thus not only non-mechanistic but also irreducible, and insofar as *(M,R)* is an accurate general model of a living system, much of modern biology therefore relies on an explanatory framework that is deemed unfit for purpose.

An attempt to prove Rosen's argument has been advanced by Louie (2005; 2007b; 2009), who has used category theory to express *(M,R)* in terms of sets of mappings, and to demonstrate that *(M,R)* contains an impredicative set, rendering it non-computable in finite time on a Turing machine (Radó, 1962; Turing, 1936; Whitehead and Russell, 1927). There is no space here to reproduce Louie's proof but, in summary, impredicativity is the condition arising when a set is a member of itself, and impredicativity may emerge in any mathematical analysis of a system that is self-referential. The individual processes within *(M,R)* are computable in finite time but, when assembled, self-reference is unavoidable and the whole *(M,R)* ceases to be computable. *(M,R)*'s irreducibility to computable software components mirrors life's irreducibility to mechanistic sub-processes.

Relational biology, in the form conceived by Rosen and Louie, has been vigorously debated (Chu and Ho, 2006; 2007a; 2007b; Goertzel, 2002; Gutierrez et al., 2011; Landauer and Bellman, 2002; Louie, 2004; 2007a; 2011; Wells, 2006), and the alleged non-computability of *(M,R)* has also inspired various attempts to instantiate it in software systems (reviewed in Zhang et al., 2016). Relational biologists do not deny that an approximation to *(M,R)*, capable of running on a Turing computer, could be created. Crucially, however, such an approximation would not capture all the properties of the *(M,R)* system. It would be merely a *simulation*, rather than a true *model*. The distinction between simulation and model is central to relational biology's critique of computational systems biology. Simulations may accurately mirror the inputs and outputs of a system, and indeed would need to do so to be judged as good simulations, but their internal causal factors – their entailment structures, in Rosen's terminology – could merely be arbitrary approximations, "black boxes" which may be pragmatically useful but essentially are the creation of the programmer. A true model, by contrast has entailment structures which logically mirror those of the real world, and correctly formed models are necessary for a genuine understanding of the system being modelled (Louie, 2009; Rosen, 1991; 2000). Weather forecasting, for instance, is largely conducted by simulation, with computers processing current weather data in the light of previous records and making a prediction for the future. Rocketry, by contrast, calculates the future position of a space satellite on the basis of data on its current physical situation and precise models derived from the laws of physics. Both may require complex calculations, but the weather forecaster does not pretend to understand, or calculate, every influence on the weather. Rocketry, by contrast, does claim a true understanding of all factors influencing the rocket's trajectory in space. Rocket science uses a model, weather forecasting uses a simulation. Relational biologists would claim that our current approach to the analysis of complex biological systems has much more in common with weather forecasting than rocket science.

In keeping with this, Louie (2011, section 2) has judged some of the software instantiations of *(M,R)* produced so far to be simulations rather than models, and this has been acknowledged by some of the authors concerned (Gatherer and

Galpin, 2013; Prideaux, 2011). Similarly, other mathematical re-workings of *(M,R)* which provide theoretical bases for computability, if not actual software instantiations (Landauer and Bellman, 2002; Mossio et al., 2009), have been likewise found lacking in various necessary aspects (Cardenas et al., 2010; Letelier et al., 2006).

Much of the controversy is dependent on Rosen's original definition of machine and mechanism (Rosen, 1964a; 1964b; 1966; 1991) which essentially stems from that of Turing (1936). However, since then, an expanded conception of the nature of machines has begun to develop, in particular the notion of X-machines (Coakley et al., 2006; Holcombe, 1988; Kefalas et al., 2003a; 2003b; Stamatopoulou et al., 2007). We believe that the current impasse over the irreducibility of *(M,R)* may be resolved by reconsidering *(M,R)* in terms of a communicating X-machine, and that is the subject of this paper.

In the Methods section we show how various formal machine architectures – namely finite state machine, stream X-machine and communicating X-machine - are conceived in abstract terms. We show how these formal architectures exist in a series – stream X-machines expanding on finite state machines, and communicating X-machines representing a further widening in scope and properties. We then repeat this process, casting *(M,R)* in terms of each formal machine architecture, pointing out the difficulties where appropriate. The stream X-machine is shown to add flexibility to the finite state machine, but nevertheless still fails to express all the properties of *(M,R)*. Then, the communicating X-machine composed of stream X-machine components is shown to be the best fit, dispensing in particular with the self-reference that is the central obstacle to computability. Finally, we discuss the kind of computer architecture necessary to implement such a formal machine architecture.

2. Methods

We follow Coakley et al. (2006) in building our communicating X-machine model through an iterative process of adding increasing levels of granularity regarding the underlying mechanistic behaviours of the system. We attempt as far as possible to reproduce the notation used in that paper, but make some small changes for two reasons: a) some of the symbols of Coakley et al. (2006) duplicate those used in *(M,R)*, in which case alternatives are introduced, b) we alter some symbols to emphasise points of similarity and difference between finite state machines and X-machines. The first step is to define the *(M,R)* system as a finite state machine (see section 2.1), before adding the concept of memory (stream X-machine; see section 2.2); and ultimately the individual instantiation, as stream X-machines in their own right, of the different system components, along with the resulting communications between them (communicating X-machines; see section 2.3) .

2.1 Finite State Machine

$$FSM = (\Sigma, Q, q_0, F, T)$$

A 5-tuple where:

- Σ is a finite alphabet of input symbols

- Q is the finite set of system states
- $q_0 \in Q$ is the initial system state
- $F \subset Q$ is a set of final (or accepting states)
- T is the transition function ($T: Q \times \Sigma \rightarrow Q$)

The transition function governs the change from one system state, $q_x \in Q$, to the next, $q_{x+1} \in Q$, according to the input received, $\sigma_x \in \Sigma$. We expand the transition function, adapting Keller (2001):

- $T = \{(T_i)_{i=1.....H}, Q, \Sigma\}$
- $q \subset Q$
- $\sigma \subset \Sigma$

$T_H(q_{H-1}, \sigma)$ is thus the final transition function in a series of H state transitions, after which the system enters state F, equivalent to q_H.

Figure 2 illustrates in graphical form the principles of the finite state machine, illustrating the interaction of current state and input within one or more functions to produce the next state in the series.

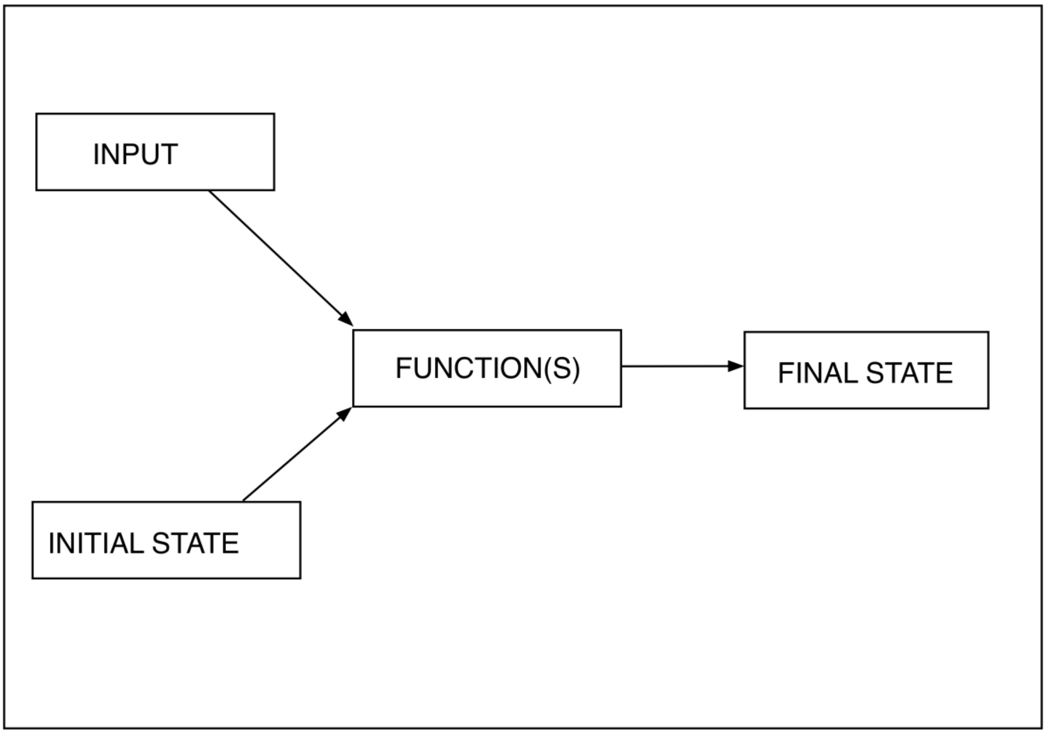

Figure 2: Finite state machine in graphical representation. Here only a single state transition is represented for clarity, but if the final state is recycled to the initial state, the process can iterate until an accepting state is reached.

2.2 Stream X-Machine

$$X = (\Sigma, \Gamma, Q, M, q_0, m_0, T, P)$$

An 8-tuple, where:

- Σ is a finite alphabet of input symbols (as for the finite state machine)
- Γ is a finite alphabet of output symbols
- Q is the finite set of system states (as for the finite state machine)
- M is an infinite set of memory states
- $q_0 \in Q$ and $m_0 \in M$ are the initial system state and initial memory state, respectively

- T is the type of the machine X, defined as a set of partial functions (T: M x $\Sigma \rightarrow$ M x Γ)
- P is the transition partial function (P: Q x T $\rightarrow$ Q)

The X-machine expands the finite state machine by virtue of the presence of stored memory states, M and output alphabet Γ. The output alphabet can be thought of as a set of signals circulating within the system or transmitted beyond the system (Stamatopoulou et al., 2007). The transition partial function of the X-machine, P, thus depends on current system state, q_x, and another partial function, T, dependent on current memory and input and which produces modified memory and output. P is therefore expressible as a 2-dimensional state transition diagram. By contrast the transition function of the finite state machine depends only on current system state and input.

Figure 3 illustrates in graphical form the principles of the stream X-machine. The "state" component is equivalent to the finite state machine (Figure 2), with the stream X-machine having an added "memory" component.

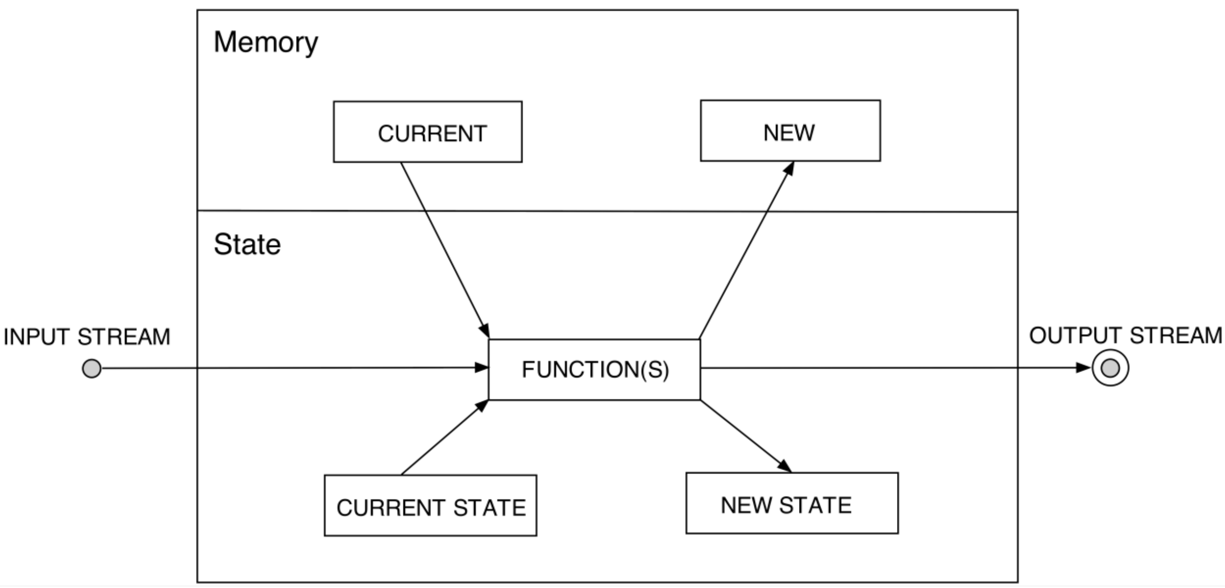

Figure 3: Stream X-machine in graphical representation. As in Figure 2, only a single state transition is represented for clarity. If the new state becomes the current state, and the new memory the current memory, the machine will iterate until an accepting state is achieved. At each iteration a new output signal is also generated.

2.3 Communicating X-Machine

Stream X-machines as defined above have no capacity to communicate with each other. Unlike finite state machines, they store memory and signal to the outside world, but have no capacity to identify and interact with other similar stream X-machines in that exterior environment. The functionality to allow communication between individual X-machines is added via a communication relation, R, as follows:

$$((C_i^x)_{i=1..n}, R)$$

Where:

- C_i^x is the *i*-th X-machine
- R is a communication relation between *n* X-machines

R is expressible as a matrix of cells(*i,j*) each defining specific communication rules between the *i*-th and *j*-th X-machine or, less prescriptively, as a list of generic communication rules that govern interaction of any X-machine with any other (Coakley et al., 2006).

Figure 4 illustrates in graphical form the principles of the communicating X-machine. The “state” and “memory” components together are equivalent to the stream X-machine (Figure 3), with the communicating X-machine having an added "communication" component consisting of a list of rules governing how the X-machines interact.

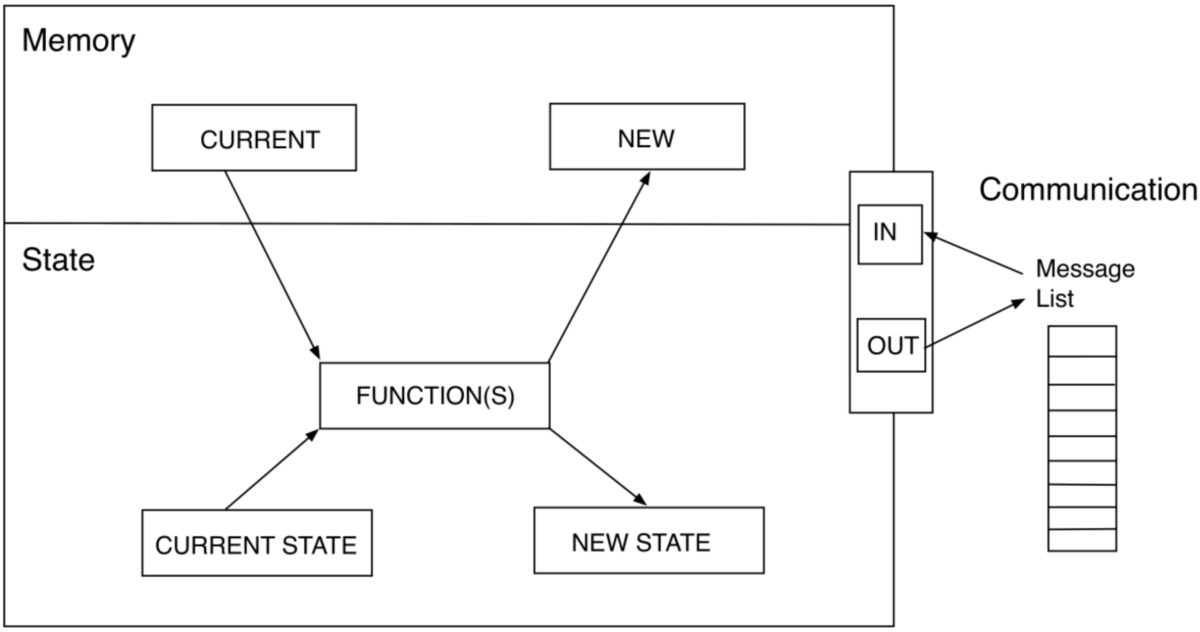

Figure 4: Communicating X-machine in graphical representation. As in Figure 3, iteration of the system via conversion of the new state to the current state, is omitted for clarity. The input-output stream of the stream X-machine is replaced by a set of communications.

3. Results

3.1 Finite State Machine

Figure 1 shows how *(M,R)* consists of three components involved in productive reactions: A, B and *f*. A is converted to B, B converted to *f* and *f* converted to φ. However, these reactions must be catalysed. In one reaction this is relatively straightforward: B→*f* requires φ. However, the other two catalysts are more complicated. B can be seen as dual-function, being the substrate for the B→*f* reaction and also the catalyst for the *f*→φ reaction. Likewise, *f* is both the substrate for the *f*→φ reaction and the catalyst for the A→B reaction. This issue has been discussed in some detail in the *(M,R)* literature (Cardenas et al., 2010; Letelier et al., 2006; Louie, 2011; Mossio et al., 2009). We therefore define *b* as the catalytic component of B, and *f′* as the catalytic component of *f*.

Mass flows within the *(M,R)* system from A to B/*b*, from B to *f*/*f′* and from *f* to φ. Our first step is therefore to attempt to express this mass flow as a finite state machine using the generic definition (Coakley et al., 2006) given in section 2.1, as follows.

Input: Σ = {*b*, *f′*, φ}

System states: Q = {A, B, *b*, *f*, *f′*, φ}

Initial system state: q_0 = {A}

Accepting states: F = {*b*, *f′*, φ}

Transition functions: T, of variants x ∈ {B, *b*, *f*, *f′*, φ} such that:

- $T = \{(T_i^x)_{i=1.....H}, Q, \Sigma\}$, specifically
  - $T_1^B = \{T: A \times f' \rightarrow B\}$
  - $T_1^b = \{T: A \times f' \rightarrow b\}$
  - $T_2^f = \{T: B \times \varphi \rightarrow f\}$

- $T_2^{f'} = \{T: B \times \varphi \rightarrow f'\}$
- $T_3^{\varphi} = \{T: f \times b \rightarrow \varphi\}$

The input set, Σ, to the finite state machine are the catalysts, which trigger the state transition functions T, but are not transformed by them. The catalysts *b* and *f′*, if defined in this way, are themselves also products of the metabolic reactions, but never substrates, hence their appearance as accepting states, F. The choice of function T_x^B over T_x^b, or T_x^f over $T_x^{f'}$, must be regarded as a stochastic choice.

The difficulties posed for finite state machines by *(M,R)* relate firstly to this necessity to enter a stochastic element into the transition process, and also to the role of catalysts in the generic state transition function T: Q x Σ → Q. T implies a separation between system state and signal, between system and environment, but catalysts are required here to be both entailments in processes, i.e. input, and also the results of those processes, i.e. system states. In Rosen's definition of a finite state machine, the entailments are all external, whereas in attempting to express *(M,R)* as a finite state machine, we require the entailments – the input signals Σ - to be states of the system itself, and for the system thereby to be self-referential. Since finite state machines cannot have this property, we therefore produce an entity which cannot be a finite state machine if it is to instantiate *(M,R)* and cannot be *(M,R)* if it is a satisfactory finite state machine.

More generally, it can also be seen that mass flow trajectories through the finite state machine as defined here will only encompass a subset of system states before reaching their accepting states. For instance, A $\rightarrow$ B $\rightarrow f \rightarrow \varphi$ does not include *f′* or *b* among the states through which it transits. Likewise, A $\rightarrow$ B $\rightarrow f'$ does not include *b* or φ, and A $\rightarrow b$ reaches an accepting state after a single state transition, and so on. Finite state machines can at best only describe sub-systems within *(M,R)*, and cannot furnish a complete description of its entirety.

3.2 Stream X- Machine

Repetition of the above exercise, expanding the finite state machine representation of *(M,R)* into a stream X-machine using the generic definition (Coakley et al., 2006) given in section 2.2, does not appreciably improve the situation. Although the stream X-machine benefits from the potential to possess memory states and generate an output alphabet, it is not clear what these properties represent in the context of *(M,R)*. For instance, memory may be used in order to allow each of the catalytic elements in the system, *b*, *f′*, φ, to be re-used, by storing a value corresponding to the number of times that catalyst operated on a substrate. If H re-uses of each catalyst were allowed, this would effectively expand the system state list to:

- Q = {A, B, $b_0 \ldots b_{H-1}$, f, $f'_0 \ldots f'_{H-1}$, $\varphi_0 \ldots \varphi_{H-1}$, Ω}

Ω is added to signify the state after the H iterations have finished. The input alphabet expands correspondingly:

- Σ = {$b_0 \ldots b_{H-1}$, $f'_0 \ldots f'_{H-1}$, $\varphi_0 \ldots \varphi_{H-1}$}

And the output alphabet is:

- $\Gamma = \{b_1 \ldots b_{H-1}, f'_1 \ldots f'_{H-1}, \varphi_1 \ldots \varphi_{H-1}, \Omega\}$

The number of accepting states reduces to:

- $F = \{\Omega\}$

We can then proceed to define the stream X-machine type, T: M x $\Sigma \rightarrow$ M x Γ, and the partial transition functions dependent on that type, P: Q x T $\rightarrow$ Q. The mappings from memory and input to memory and output constituting the type, T, are best visualised in tabular form (Table 1). Memory, M, is defined as a variable that allows for H re-uses of each catalyst prior to the accepting state Ω.

		Σ		
		b_n	f'_n	φ_n
M	0	b_1+M_1	f'_1+M_1	φ_1+M_1
	n	$b_{n+1}+M_{n+1}$	$f'_{n+1}+M_{n+1}$	$\varphi_{n+1}+M_{n+1}$
	H	$\Omega+M_0$	$\Omega+M_0$	$\Omega+M_0$

Table 1: T-functions for the stream X-machine realization of *(M,R)*. Rows M define memory states over n = zero to H. Columns Σ define the inputs also over n = zero to H-1. Table values define the output and next memory state.

Table 1 illustrates the re-use of catalytic elements for H occasions. Each time a catalyst is used, the memory state of the system is ratcheted up by one, and the catalyst re-emerges as output. On the H^{th} occasion the system dies, Ω is returned and memory is reset to zero. Table 1, representing T: M x $\Sigma \rightarrow$ M x Γ, can then be combined with system states in the state transition diagram, P: Q X T $\rightarrow$ Q (Table 2)

		Q					
		A	B	$b_0 \ldots b_{H-1}$	f	$f'_0 \ldots f'_{H-1}$	$\varphi_0 \ldots \varphi_{H-1}$
T	$b_x M_x$				φ		
	$f'_x M_x$	B/b					
	$\varphi_x M_x$		f/f'				

Table 2: P-functions for the stream X-machine realization of *(M,R)*. Columns Q define system states. Rows T define the T-functions (Table 1), over x=1 to x=H-1. Table values define the next system state. Empty cells indicate invalid Q/T combinations, thus generating null returns on system state.

The rows of Table 2, T, are a compaction of Table 1, representing each combination of input Σ and memory M at time x and how it interacts with the set of system states, Q, to produce a new system state. Table 2 is a sparse state transition diagram as $\{b_0 \ldots b_{H-1}, f'_0 \ldots f'_{H-1}, \varphi_0 \ldots \varphi_{H-1}\} \subset$ Q do not generate state transitions. As with the transition functions of the finite state machine (Section 3.1), the partial functions acting on A and B will produce either B or *b*, or *f* or *f'*, respectively with stochastic distribution of probabilities. Expansion of the finite state machine to a stream X-machine therefore does not immediately suggest a solution to the problems of defining entailment and state, or of self-reference, and therefore again falls short of a mechanistic realization of *(M,R)*.

3.3 Communicating X-Machine

Communicating X-machines (section 2.3) build upon the concept of stream X-machines so that they may be used to model at the component or sub-system level, and allow communication between these individual components/sub-systems to facilitate emergent behaviour at the level of the entire system. As such, communicating X-machine systems are comprised of multiple instantiations of the different types of stream X-machine components. For *(M,R)*, their interactions may be abstractly represented in matrix form (Table 3):

		i					
		A	B	b_x	f	f'_x	φ_x
j	A					$\mathrm{B}/b+f'_{x+1}$	
	B						$f/f'+\varphi_{x-1}$
	b_x				$\varphi+b_{x+1}$		
	f			$\varphi+b_{x+1}$			
	f'_x	$\mathrm{B}/b+f'_{x+1}$					
	φ_x		$f/f'+\varphi_{x+1}$				

Table 3: Communication relations, R, between the i^{th} and j^{th} stream X-machines in a communicating X-machine. Entries describe the system states of the i^{th} and j^{th} stream X-machines after each interaction. Empty cells indicate non-interacting combinations, thus generating null returns on system states.

Unlike Table 2, which shows state/memory transitions within a single stream X-machine, Table 3 shows the rules governing the interaction of two stream X-machines. The entailments are thus external to each stream X-machine but internal to the communicating X-machine of the entire system. Table 3 only presents the consequences of communication between two stream X-machines in terms of their system states. Their memory states and other internal properties will alter as described in section 3:2. Table 3 assumes that the memory value, x, can increase indefinitely, but where x = H, states f'_{x+1} , b_{x+1} and φ_{x+1} will be Ω.

Crucially, there is no self-reference represented within Table 3. The entailments operating on each individual stream X-machine are external, i.e. emanate from other stream X-machines. An individual stream X-machine will not undergo a state transition unless it encounters another stream X-machine that can deliver the appropriate signal.

3.4 Object-Oriented Communicating X-Machine

We previously attempted (Zhang et al., 2016) to represent *(M,R)* using Unified Modelling Language (UML) which provides various tools for object-oriented systems analysis. Correctly formed UML constitutes a basis for representation of the modelled system in any object-oriented programming language. Using UML, we were able to construct UML state machine diagrams for individual classes in *(M,R)*, where A, B, *b*, *f*, *f′* and φ are classes composed of objects of that type (Figure 6 of Zhang et al. (2016)). We also constructed a UML communication diagram (Figures 4 and 5 of Zhang et al. (2016)) which we noted bore a strong resemblance to Rosen's original *(M,R)* diagram. The UML communication diagram is conceptually equivalent to the communication relations matrix, R, presented here in Table 3. To attempt to synthesise the communicating X-machine and object-oriented approaches to *(M,R)*, we begin with the cartoon diagram of Figure 5, which illustrates an *(M,R)* system, arbitrarily bounded for clarity, populated by a selection of the relevant objects using a simplified UML class notation.

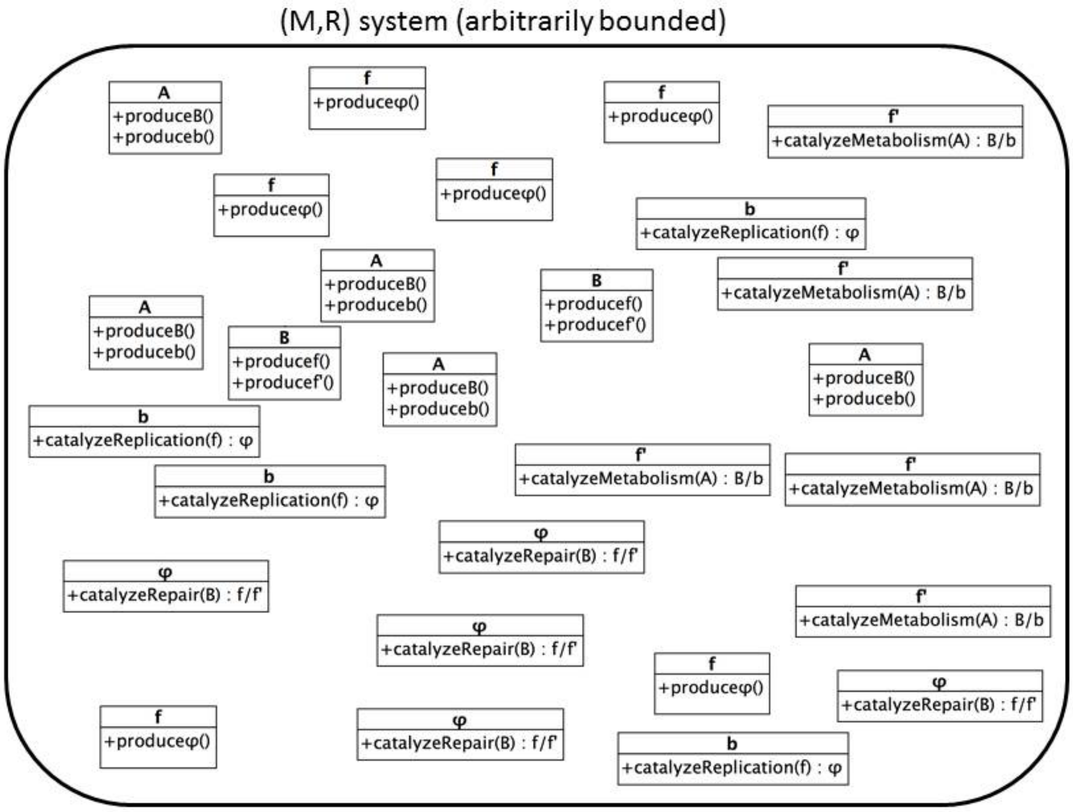

Figure 5: Object-oriented *(M,R)* instantiation. Objects of the six classes A, B, *b*, *f*, *f'* and φ as defined by Zhang et al. (2016) contained within an arbitrary system boundary.

Each of the objects within Figure 5 is represented in the simplified UML class notation with its functions below the horizontal line. For instance, an object of class *f* has a function +produceφ(), indicating that this object can be transformed into an object of class φ, which will then possess the function +catalyseRepair(B): *f/f'*, indicating that it will catalyse the production of *f* or *f'*, by stochastic choice previously discussed, from B. Representing the objects as individual communicating X-machines, with all of the associated syntax for inputs, memory, states, functions and outputs (not shown), resulted in an overwhelmingly complicated diagrammatic model. As such, we have developed the cartoon diagram in Figure 6, which integrates the object-oriented *(M,R)* diagram in Figure 5 with the communication relations matrix in Table 3, and also adds a memory component (as in Figure 4) to those objects that require it.

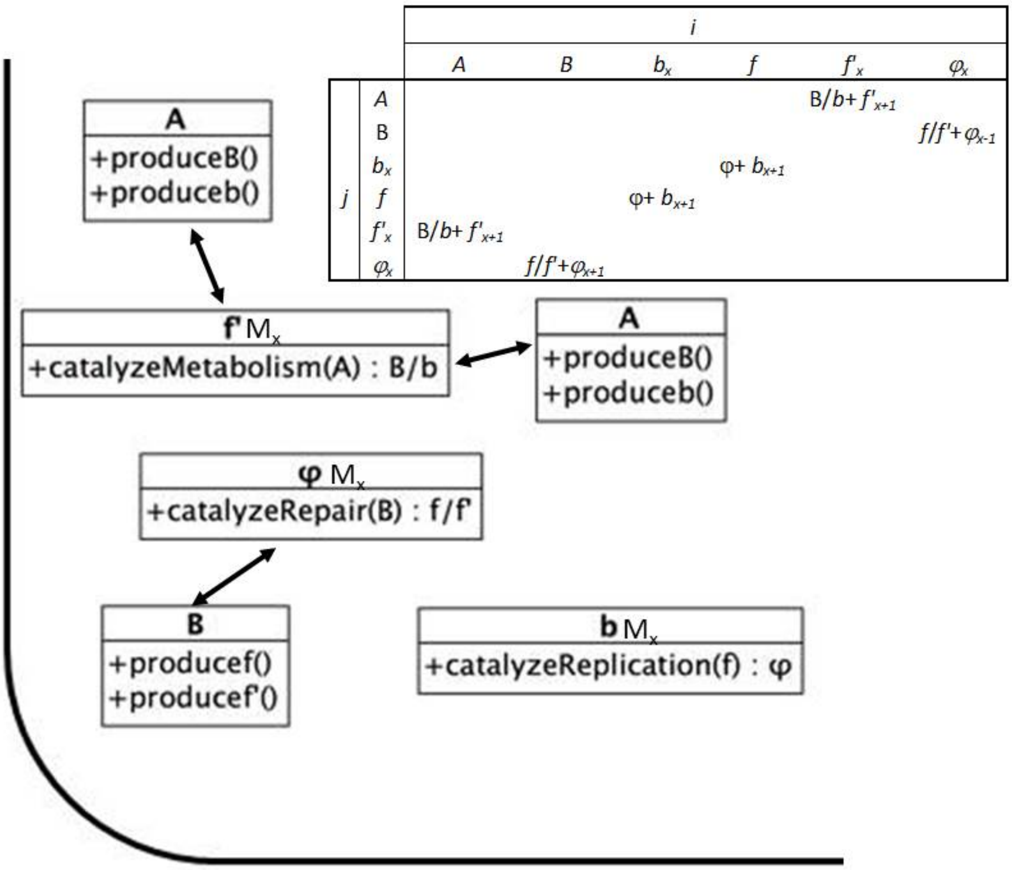

Figure 6: Object-oriented *(M,R)* instantiation as communicating X-machine. Detail of Figure 5, with the addition of the communication relations matrix, R, (Table 3) as inset. Arrows indicate interactions as specified by R.

In Figure 6, each object is connected by a double-headed arrow to each other object with which it is capable of communication, as specified by the communications relations matrix, R. Notice that the object of class *b* does not have any communication relation within this frame, since it can only interact with objects of class *f* - not shown in Figure 6 simply for reasons of space. Figure 6 differs from Figure 5 in that each object has its memory state added in the form M_x, following Table 1. This extends the original class diagrams given in Figure 2 of Zhang et al. (2016). M_x corresponds to the memory component of the communicating X-machine (Figure 4).

This concludes our presentation of *(M,R)* as three formal machine architectures. The first of these, the finite state machine, cannot capture self-reference and therefore obviously fails to instantiate *(M,R)*. The second, the stream X-machine, permits some additional detail to be added to the system in terms of memory states, which assists with issues such as the number of times a catalyst can be reused, but nevertheless does not solve the problem of self-reference. Only the third formal architecture, the communicating X-machine, allows us to transcend this impasse. It does so by treating each component of *(M,R)*, rather than the entire system, as a stream X-machine, and then forcing all entailments to be between individual stream X-machines in the form of messages. The problem of self-reference, and the consequent mathematical impredicativity and Turing non-computability that is the central argument of relation biology as conceived by Rosen and Louie, is therefore sidestepped. Object-orientation is a useful framework within which to build the *(M,R)* communicating X-machine.

4. Discussion

One of Rosen's early papers on *(M,R)* (Rosen, 1964a) involved the analysis of *(M,R)* systems as sequential machines (Ginsburg, 1962), very close to finite state machines as defined in section 2.1. Comparing the two, he remarked (pp. 109-110 of that paper):

"in the theory of sequential machines [.....] it is generally possible to extend the input alphabet without enlarging the set of states: that we cannot do […] directly in the theory of *(M,R)*-systems [which] points to a fundamental difference between the two theories."

This is essentially the same conclusion we draw in section 3.1 – in *(M,R)*, states and input cannot be separated, thus making instantiation of *(M,R)* as a finite state machine impossible. Expansion of the finite state machine to a stream X-machine is also inadequate, as the same problem of disentangling entailments from system states remains despite the addition of memory and output signalling functions. Generally, finite state machines and stream X-machines are designed at the system-level, and are therefore abstractions of machines that receive their entailments from the environment. *(M,R)*, by virtue of its entirely internal entailment relations and consequent self-referential nature, cannot fit either simple finite state machine or stream X-machine requirements. A machine that adequately represented *(M,R)* would require the capacity to be in two states simultaneously, or to have no states at all - in Rosen's own words, to have "entailment without states" (Rosen, 1991). Since both of these defy our common-sense logic concerning machines, this would seem to re-inforce the general refutation of mechanism in biology that stems from Rosen's work on *(M,R)*.

However, this conclusion rests on two premises:

1) *(M,R)* is represented as a single machine.
2) That machine representation of *(M,R)* is processed sequentially.

Communicating X-machines are by definition composites of individual stream X-machines. For a communicating X-machine model composed of *n* stream X-machines with memory maximum H, each stream X-machine may have states:

- $Q = \{A, B, b_0 \ldots b_{H-1}, f, f'_0 \ldots f'_{H-1}, \varphi_0 \ldots \varphi_{H-1}, \Omega\}$

as outlined in section 3.2, producing a total of 3H+4 possible states for each stream X-machine and a total state space, $\mathcal{Q}$, of $n^{(3H+4)}$ for the communicating X-machine. For $n = 100$ and H = 3, $\mathcal{Q} = 10^{26}$. Exhaustive permutation of the entire state space of the communicating X-machine therefore runs into technical problems - a single processor at 10^{10} FLOPS would require 10^{16} seconds, or 3.17×10^{8} years to traverse all the possibilities. Parallel processing is thus required, both from a standpoint of computational tractability, and arguably also because parallel activity is intuitively more in keeping with the nature of living systems (see Gatherer, 2007; Gatherer, 2010 for further exploration of this issue).

The communicating X-machine paradigm is therefore of necessity a massively parallel machine architecture, composed of individual stream X-machines, that permits all entailments to be internal to the system as a whole, but where for each individual X-machine within that system, the entailments are external, i.e. they are transmitted as communications from other stream X-machines in the collective. Each component stream X-machine at any moment has a system state which can also represent an entailment for any other component stream X-machine that it encounters within the system. The communicating X-machine paradigm is the only formal machine architecture that is capable of representing *(M,R)*. Rosen's insistence that *(M,R)* cannot be instantiated as a machine on account of its circular entailment structures and the paradoxes that arose from attempting to impose states onto it – which led to Rosen's statement that *(M,R)* is state-free – can be seen to be consequences of a limited definition of a machine. The use of the communicating X-machine architecture also deals with problems arising in our previous (Zhang et al., 2016) object-oriented analysis of *(M,R)*, for instance our inability to produce a convincing UML state machine diagram for the entire system. We were, however, able to produce UML state machine diagrams for individual classes of objects, and these could provide the basis for their treatment as individual stream X-machines within a communicating X-machine environment. The communicating X-machine provides the missing element in our object-oriented analysis of *(M,R)*.

Some problems nevertheless remain. As with our previous attempted practical instantiation of *(M,R)* in process algebra (Gatherer and Galpin, 2013), this theoretical instantiation as a communicating X-machine forces us to take a literal stance towards the Goudsmit (2007) representation of *(M,R)* (Figure 1). A, B, *f* and φ are no longer interpretable as general descriptions of metabolic or replacement functions but are sets of interacting molecules and the arrows within the *(M,R)* diagram represent events happening to such individual molecules. Also, we are still faced with the problem of how dual-function components of *(M,R)* are to be defined within the system. The relation between B as substrate and *b* as catalyst has been the subject of much discussion (Cardenas et al., 2010; Letelier et al., 2006; Louie, 2011; Mossio et al., 2009), mainly because it is poorly defined with the relational biology literature stemming from Rosen and his disciples. If we have not answered this issue it is because we are still unsure of the question. The resulting compromise, used by us here and previously (Gatherer and Galpin, 2013; Zhang et al., 2016), is simply to allow a stochastic choice of catalytic or substrate product for the A→B and B→*f* reactions. For some this may be a fatal flaw, but we submit that living systems are stochastic to some extent.

The communicating X-machine paradigm expands the definition of a machine to something massively parallel, complex yet self-contained. It is a more life-like machine than the limited definitions of the 20th century. *(M,R)* was not one of those old machines, but something else entirely. Rosen's error was to conclude that it could not be a machine of any kind. We can now see what kind of a machine it is. It is also reducible. Understanding of the properties of the individual stream X-machines does lead to an understanding of the whole system through its representation as a communicating X-machine. Systems biology may yet turn out to be both mechanist and reductionist.

Acknowledgements and Data Access Statement

No raw data were generated in the course of this project. We thank numerous colleagues who have critiqued this paper, without of course implying their joint responsibility for any failings it may have. MLP performed this work as part of the requirements for an MSci degree at Lancaster University.

5. References

Cardenas, M.L., Letelier, J.C., Gutierrez, C., Cornish-Bowden, A., and Soto-Andrade, J., 2010. Closure to efficient causation, computability and artificial life. Journal of Theoretical Biology 263, 79-92.

Carnap, R., 1934. The Unity of Science. Thoemmes Press, Bristol, 1995.

Chu, D., and Ho, W.K., 2006. A category theoretical argument against the possibility of artificial life: Robert Rosen's central proof revisited. Artificial Life 12, 117-134.

Chu, D., and Ho, W.K., 2007a. Computational realizations of living systems. Artificial Life 13, 369-81.

Chu, D., and Ho, W.K., 2007b. The localization hypothesis and machines. Artificial Life 13.

Coakley, S., Smallwood, R., and Holcombe, M., Using X-machines as a formal basis for describing agents in agent-based modelling, in: Hamilton, J. A., et al., Eds.), Proceedings of the 2006 Spring Simulation Multiconference (SpringSim' 06), The Society for Modeling and Simulation International, San Diego, Calif. 2006, pp. 33-40.

Comte, A., 1830. Cours de philosophie positive. Bachelier, Paris,.

Crick, F., 1966. Of molecules and men. University of Washington Press, Seattle,.

Crick, F., 1981. Life itself : its origin and nature. Simon and Schuster, New York.

Elsasser, W.M., 1998. Reflections on a Theory of Organisms. Holism in Biology. The Johns Hopkins University Press, Baltimore.

Gatherer, D., 2007. Less is more: the battle of Moore's Law against Bremermann's Limit on the field of systems biology. BMC Syst Biol 1 supp.1, 53.

Gatherer, D., 2010. So what do we really mean when we say that systems biology is holistic? BMC Systems Biology 4, 22.

Gatherer, D., and Galpin, V., 2013. Rosen's (M,R) system in process algebra. BMC Systems Biology 7, 128.

Ginsburg, S., 1962. An introduction to mathematical machine theory. Addison-Wesley, Reading, Mass.

Goertzel, B., Appendix 2. Goertzel versus Rosen: Contrasting views on the autopoietic nature of life and mind., Creating Internet Intelligence, Kluwer Academic/Plenum Publishers, New York 2002.

Goudsmit, A.L., 2007. Some reflections on Rosen's conceptions of semantics and finality. Chemistry and Biodiversity 4, 2427-2435.

Gutierrez, C., Jaramillo, S., and Soto-Andrade, J., 2011. Some thoughts on A.H. Louie's "More Than Life Itself: A Reflection on Formal Systems and Biology". Axiomathes 21, 439-454.

Holcombe, M., 1988. X-Machines as a Basis for Dynamic System Specification. Software Engineering Journal 3, 69-76.

Kefalas, P., Eleftherakis, G., and Kehris, E., 2003a. Communicating X-machines: a practical approach for formal and modular specification of large systems. Information and Software Technology 45, 269-280.

Kefalas, P., Eleftherakis, G., and Kehris, E., 2003b. Communicating X-machines: From theory to practice. Advances in Informatics 2563, 316-335.

Keller, R.M., Computer Science: Abstraction to Implementation, Available: http://www.cs.hmc.edu/~keller/cs60book/%20%20%20Title.pdf, Harvey Mudd College 2001.

Landauer, C., and Bellman, K., 2002. Theoretical biology: Organisms and mechanisms. AIP Conference Proceedings 627, 59-70.

Lenzer, G., 1998. Auguste Comte and positivism : the essential writings. Transaction Publishers, New Brunswick, NJ.

Letelier, J.C., Cardenas, M.L., and Cornish-Bowden, A., 2011. From L'Homme Machine to metabolic closure: Steps towards understanding life. Journal of Theoretical Biology 286, 100-13.

Letelier, J.C., Soto-Andrade, J., Guinez Abarzua, F., Cornish-Bowden, A., and Cardenas, M.L., 2006. Organizational invariance and metabolic closure: analysis in terms of (M,R) systems. Journal of Theoretical Biology 238, 949-61.

Loeb, J., 1912. The Mechanistic Conception of Life. University of Chicago Press, Chicago.

Louie, A.H., Rosen 1, Goertzel 0: Comments on the appendix “Goertzel versus Rosen”, Available: http://panmere.com/rosen/Louie%20-%20GoetzelvsRosen.pdf, Vol. 2015. 2004.

Louie, A.H., 2005. Any material realization of the (M,R)-systems must have noncomputable models. Journal of Integrative Neuroscience 4, 423-36.

Louie, A.H., 2007a. A living system must have noncomputable models. Artificial Life 13, 293-7.

Louie, A.H., 2007b. A Rosen etymology. Chemistry and Biodiversity 4, 2296-314.

Louie, A.H., 2009. More than Life Itself. A Synthetic Continuation in Relational Biology. Ontos Verlag, Frankfurt.

Louie, A.H., 2011. Essays on More Than Life Itself Axiomathes 21, 473-489.

Mazzocchi, F., 2012. Complexity and the reductionism-holism debate in systems biology. Wiley Interdiscip Rev Syst Biol Med 4, 413-27.

Molière, 1673. The Imaginary invalid ... Translated by Bert Briscoe, 1967. [With plates.]. C. Combridge, Birmingham.

Monod, J., 1971. Chance and necessity; an essay on the natural philosophy of modern biology. Knopf, New York,.

Mossio, M., Longo, G., and Stewart, J., 2009. An expression of closure to efficient causation in terms λ-calculus. Journal of Theoretical Biology 257, 489-498.

Polanyi, M., 1968. Life's irreducible structure: Live mechanisms and information in DNA are boundary conditions with a sequence of boundaries above them. Science 160, 1308-1312.

Prideaux, J.A., 2011. Kinetic models of (M,R)-systems. Axiomathes 21, 373-392.

Radó, T., 1962. On non-computable functions. Bell System Technical Journal 41, 877–884.

Rashevsky, N., A unified approach to physics, biology and sociology, in: Rosen, R., (Ed.), Foundations of Mathematical Biology. 3: Supercellular Systems, Vol. 3. Academic Press, New York 1973, pp. 177-190.

Rosen, R., 1964a. Abstract Biological Systems as Sequential Machines. Bulletin of Mathematical Biophysics 26, 103-11.

Rosen, R., 1964b. Abstract Biological Systems as Sequential Machines. Ii. Strong Connectedness and Reversibility. Bulletin of Mathematical Biophysics 26, 239-46.

Rosen, R., 1966. Abstract biological systems as sequential machines. 3. Some algebraic aspects. Bulletin of Mathematical Biophysics 28, 141-8.

Rosen, R., 1991. Life Itself: A Comprehensive Inquiry into the Nature, Origin, and Fabrication of Life. Columbia University Press, New York.

Rosen, R., 2000. Essays on Life Itself. Columbia University Press, New York.

Stamatopoulou, I., Kefalas, P., and Gheorghe, M., 2007. Modelling the dynamic structure of biological state-based systems. Biosystems 87, 142-149.

Turing, A.M., 1936. On computable numbers, with an application to the Entscheidungsproblem. Proc. London Math. Soc. 42, 230-265.

Waddington, C.H., The basic ideas of biology, in: Waddington, C., (Ed.), Towards a Theoretical Biology. 1. Prolegomena, Vol. 1. Edinburgh University Press, Edinburgh 1968, pp. 167-173.

Wells, A.J., 2006. In defense of mechanism. Ecological Psychology 18, 39-65.

Whitehead, A.N., and Russell, B., 1927. Principia Mathematica. Cambridge University Press, Cambridge, 1963.

Zhang, L., Williams, R.A., and Gatherer, D., 2016. Rosen's (M,R) system in Unified Modelling Language. Biosystems 139, 29-36.