Specification of the Stringmol chemical programming language version 0.2

Technical Report Number YCS-2010-458

Simon Hickinbotham[1], Edward Clark[1], Susan Stepney[1], Tim Clarke[2], Adam Nellis[1], Mungo Pay[2], Peter Young[3]

[1]Department of Computer Science, [2]Department of Electronics, [2]Department of Biology
York Centre for Complex Systems Analysis, University of York, UK

sjh@cs.york.ac.uk www.yccsa.org

This work was supported by EPSRC grant EP/F031033/1

Abstract

This report describes in detail version 0.2 of a specification for a nonstandard computational model that emulates the reactivity of enzymes in bacterial cells. The model is an artificial chemistry in which the unique properties of each molecular species is encoded as a sequence of symbols. The model has a very simple cell-level representation consisting of a mixing volume and mixing equation. There is no distinction between genotype and phenotype, or data and program. Randomly selected molecules are subject to a stochastic binding test, in which their identifying sequences are subjected to a complementary alignment process. If the bind is successful, the molecules are combined to form a computational entity to run the reaction between them. At this point the sequence acts as a microprogram. These microprograms can perform a range of tasks, much as a chemical reaction can. Our first chemistry within this model solves the problem of designing a chemical "replicase", capable of creating copies of itself such that it can replenish a population of molecules that are subject to a decay process. This report gives detail to the model, describes the replicase molecule and its function, and shows an "invasion when rare" experiment.

---

1 Introduction

The traditional computational genetic algorithm that has existed since the 1960s is a very loose analogy with biological evolution. Much research effort has been expended in refining the process, but the basic model has remained relatively unchanged. We propose that a richer evolutionary model can deliver more powerful evolutionary processes. We seek alternative formulations of genetic algorithms in which complex genotype-phenotype (geno-pheno) interactions are used to organise the genome, control bloat, and thus deliver efficient learning systems. The challenge is to define a model that is *sufficiently rich* to allow such processes to arise.

Bacteria are the principal inspiration for our model, since their gene expression mechanisms are relatively well understood, their ontogeny is relatively simple, and they use horizontal gene transfer to spread a rapid response to new environmental conditions through the population. A key challenge is to create a metabolic model which is computationally efficient, yet allows the richness of geno-pheno interaction that we require. We believe that a fully functional bacterial model would be too difficult to devise *ab initio*. Instead, we concern ourselves with developing molecular analogues that have sufficient functionality to allow rich interactions between them to emerge. The model must encode metabolites and genomic material in such a way that they can interact. In this light, an artificial chemistry is an attractive solution, since it maintains an intuitive link with real biochemical systems, albeit in a simplified form. In addition, we limit the set of legal molecular-level events to simplify the design process.

In the early stages of this project [1] , we have concentrated on modelling bacterial processes via a limited series of reaction rules [1] and have applied such rule-based models to robot control [2]. We have designed a computational metabolome and implemented it as a network of reactions between a maximum of two molecular agents. This early work demonstrated that with simple computational restrictions on the types of reaction that are permitted, it is straightforward to design a gene regulatory network. To make the reaction networks evolvable, a system must be able to subtly change the nodes (substrates), edges (reactions), and rates in the network, via changes in the binding, reaction and decay of the molecular structures. Given that the biological systems we are emulating are far to complex to simulate in full detail, our abstractions need to find appropriate granularity of change in the expressed proteins with the characteristics of mutation that

[1]This report forms part of the Plazzmid project, EPSRC grant EP/F031033/1. Later versions of this report will incorporate revisions to the molecular specification.

the system possesses. Thus we must develop an artificial chemistry (AC) as the basis for the geno-pheno interaction.

We are investigating a range of ACs. There are three broad classes: abstract (molecular properties specified directly, so no analogy with shape) [3]; spatial shape-based [4]; program-based. Here we explore the program-based approach, with reference to a set of biological principles that we believe characterise biological systems. There are four components to this model:

1. A very simple cell-level architecture, more analogous to the containing vessels in Spiegelman's [5] RNA experiments than living cells, but which allows reactive agents to mix.
2. "RNA world", in which there is no distinction between genotype and phenotype - the machine acts as the blueprint for itself.
3. A "soft" binding process, based on Smith-Waterman alignments.
4. An analogue of protein folding, using computational program flow control (like AVIDA and Ray's Tierra [6, 7]) to allow reactions to be richly encoded.

We are working towards an agent-based system, where each agent is capable of emulating as many of the different molecular species seen in biology as possible: substrates, products, enzymes, proteins, RNA and DNA. We have designed a computational emulation of the biological system just described. All molecules in our system are defined in terms of a sequence of symbols. The symbols encode a set of bindings between symbols, and low-level programming instructions. A combination of the properties of these instructions and the sequence they are assembled into forms the basis of a "microprogram" that can emulate the range of protein functions we see in nature. The mixing of these molecules in a stochastic chemistry allows the overall program of the system to emerge. Key to this approach is the use of a sequence alignment algorithm for binding between individual molecules and for branching of molecular microprograms. The inexact alignments of sequences allows rates of binding and execution of the microprograms to evolve. This report gives full detail to the microcodes, their execution, and the use of sequence alignment to implement an evolvable artificial chemistry.

We have developed the artificial chemistry with a specific design challenge in mind: Can a system be built that consists of reactive agents analogous to proteins, whose function is determined by a sequence of symbols, and which are capable of maintaining their population by constructing copies of themselves by reference to the symbol sequence? We call these agents "replicase molecules", since their function is to replicate themselves. This system has the advantage that only one kind of molecule needs to be present, but the function of the molecule captures the level of geno-pheno interaction that we will require in our bacterial model that is to follow.

2 Bioinspiration for the Stringmol domain model

2.1 The Cell model

Our cell model need not be complex, since the focus on this paper is molecular interactions. Speigelman [5] showed that evolvable systems can be run *in vitro* given appropriate raw materials. We follow this methodology for our current work *in silico*: the cell is little more than a container in which energy and raw materials are available for the evolvable reaction. The volume of the container is important since it can be used to control the probability of one molecular agent encountering another.

2.2 RNA-world chemistry

The RNA-world hypothesis proposes that early life on earth was formed from RNA, which is capable of storing information in a similar fashion to DNA and acting as an enzyme in a similar fashion to proteins.

Our molecules are similar to the RNA-world model, since there is no distinction between genotype and phenotype - the machine (phenotype) is also the blueprint (genotype) for making copies of itself. In a reaction, each replicase molecule can act as either "copied" or "copier" when it encounters another molecule. The function of an RNA molecule is determined by its sequence. Regions of the sequence act as binding sites for other molecules, and other regions determine the reaction that the molecule might be involved in. Thus our model differs to modern biology, in which DNA and RNA sequences are composed of different subunits (nucleotides) from proteins (amino acids). Here, we want to explore how a sequence of codes can capture a desired function within a larger biological system. We do not currently concern ourselves with the translation of information between nucleotides and amino acids.

2.3 Inexact subsequence alignment

Copying is a bi-molecular process in which one partner acts as the template and one acts as the copier. In order to be copied, a replicase molecule must possess a sequence that matches a sequence on the partner replicase, and thus facilitate *binding* to it. Our earlier work [8] shows that the *rate* of binding is important in tuning a gene regulatory network. Our goal is to derive the bind probability from some similarity score between sequences. We do not wish to develop new algorithms to measure this similarity. Fortunately, a suitable subsequence alignment procedure is available to us. Smith-Waterman alignment [9] is the basis of comparison of genetic sequences to determine phylogeny between species. It is designed to detect non-random sequence matches and give a score to any alignment found. Gaps and mismatches can be accommodated for, and the penalty for each can be tuned as desired. This algorithm forms the basis of our molecule binding procedure, here applied to complementary alignments.

2.4 Protein folding and stack-based programming languages

Biological genome maintenance is carried out by proteins (folded sequences of amino acids) and small RNA molecules (sRNAs), both of which are built following a specification on the genome. The functional characteristics of the protein arise from its folded shape, which depends on the chemical properties of the amino acids. The sequence of amino acids forms the primary (1°) structure of the protein, but the process of folding into the tertiary (3°) structure is not explicitly encoded on the DNA, emerging from the chemical properties of the amino acids themselves. The functionality of the protein is thus a product of the DNA-based specification of the amino-acid sequence, the physical and chemical properties of the amino acids, and the physical and chemical properties of the protein structure itself. There is a high degree of interdependency between these factors. Proteins' folded shapes are highly structured and often flexible. Reactions occurring between proteins and other molecular entities cause changes in the tertiary structure of the protein, which in turn change the reactive properties of the protein. Sometimes the shape of the protein is the most significant factor. For example, proteins can form loops that can clamp onto DNA, physically obstructing other proteins from interacting with it. At other times, the charge on the protein's surface changes the conformation of its neighbours. Over evolutionary time scales, the genetic blueprint develops a deep (but implicit) interaction with these properties.

Protein folding (and indeed RNA folding) are difficult processes to emulate (beyond the current state of the art [10]). However, faithful rendering of molecular shape is not explicitly required for the binding scheme we are developing - we simply require some inexact matching process that emulates the effect that "docking" has on the chance of molecules coming together to react. If we abandon a shape-based representation, we will also require a means of deriving the function of a molecule from its sequence alone without some shape-based process. Fortunately, there is preexisting work in this area also, in the form of stack-based programming languages where folding is represented abstractly by the concept of program flow. There is a rich history of ALife based on the concept of individual-as-program. Key examples are Tierra [7], AVIDA [11], nanopond [12] and "BF" [13]. All these place heavy emphasis on the genotype: each "organism" has a template code, plus some registers and some energy, which together confer function and

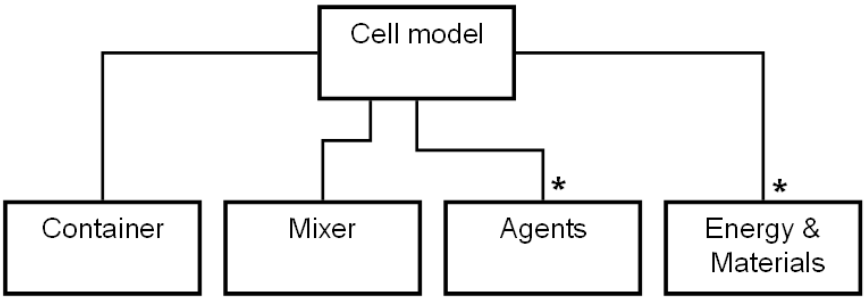


Figure 1: Components of a simple cell model

fitness on the individual. The phenotype is forced to be simple: essentially a sequence of instructions, plus a miniature processing architecture that executes the genetic instructions. In these systems the processing architecture of each organism is not subject to heritable change. We argue that the ALife organism needs to be richer. The gene sequence must encode as much of the function of the organism as possible. As a first step towards this goal, we use executing sequences as the basis for the biochemistry of the organism, rather than as a representation of the entire organism in one unit. By stating that an organism is composed of a *set* of simple executing sub-programs, we can make simpler sub-units, draw a closer analogy with functioning proteins, and make more of the processing machinery available to evolve.

2.5 Summary

We have listed above the biological phenomena that have led us to the design of our molecular model. This model consists of a space in which computational agents mix, bind and react with each other. The mixing is an external process, but the binding rate, and the nature of the reaction are encoded on the sequence of instructions that define each molecular species. Having given background to the processes that have led us to develop this model, we now turn to the description of the model itself.

3 Cell-level architecture

This section enlarges upon the description of the domain model in [8]. The metabolic model is composed of four components, as shown in figure 1.

Firstly, there exists a *container*, which specifies the volume and dimensionality of the space in which the agents exist. We specify a simple 2D container of area v_c = 2500 units.

Secondly, we have a set of metabolite *agents*, of area v_a = 10 units which are present in varying quantities in the container, analogous to the various quantities of different molecular species in a biological system. Agents are considered to be *bound* or *unbound*. If an agent is bound to another agent, then one member of the pair is executing its program (it is *active*), whereas the other member of the pair is *passive*. The program that the pairing executes is encoded in the sequences of the bound agent pair. We refer to this as the *Reaction*.

Thirdly we have a stochastic *mixer*, which governs the movement and changes in adjacency of the elements within the container. For a bimolecular reaction such as the bind B, our mixer utilises a simple propensity function P(B), which estimates the probability of two agents being sufficiently close enough for the reaction to occur. For any one agent in a bimolecular reaction, the chance of the second agent in the reaction being close enough to react is:

$$\mathrm{P}(B|v_c, v_a, n) = 1 - (1 - (v_a/v_c))^n \quad (1)$$

Reaction	Rule format		
Binding:	Agent + Agent	→	Reaction
Production:	Reaction	→	Reaction + Agent
Dissociation:	Reaction	→	Agent + Agent
Decay:	Agent	→	Ø

Table 1: The four types of reaction rule in the metabolic controller

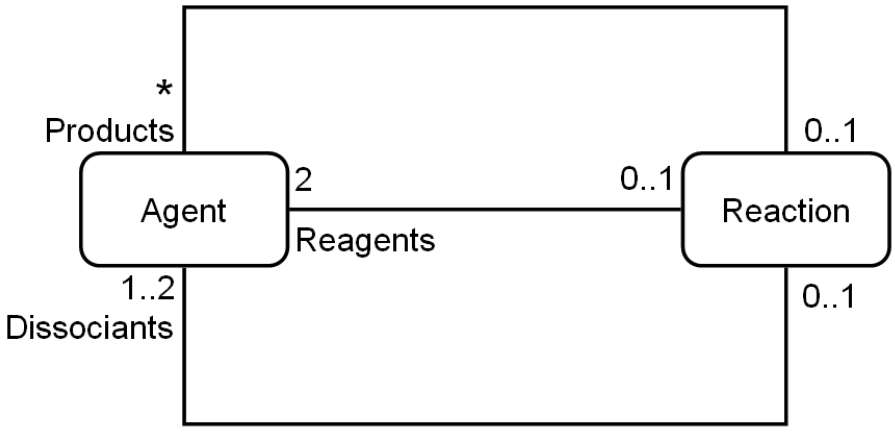


Figure 2: Class diagram for Agents and Reactions.

where n is the number of instances of the second agent in the metabolism. Space in the system is represented abstractly via the ratio of container area to agent area. Apart from this consideration, the model is aspatial.

Fourthly, energy and raw materials are available to the agents as they react. These are minor components to the model. Energy is supplied to the system at a fixed rate per time step. The only raw materials in the system are the symbols used in construction of the string. These are assumed to be present at saturation throughout the experiment, although conservation of mass could be implemented in the system straightforwardly. The state of the system is recorded in a series of discrete time steps. Each timestep is processed via examination of all agents in a random sequence. One binding attempt or one execution of an instruction requires one unit of energy. Energy not used in one time step is available in the next. Energy is modelled as a continuum since the computational cost of representing energy via individual particles is too great. This is an appropriate abstraction for the problem domain, and is in line with current practice in systems biology [14]. Waste materials, and materials required for construction of agents are assumed to be approximately constant and are not represented in the current model.

4 Reactions between agents

In earlier work [1] , we have shown that a rich regulatory system can be built following reactions with no more than two reactants and no more than two products. There, we specified the reactions in a metabolism via a simple reaction table. Having shown that a rich system can be created when using a simple set of reaction rules, we can now extend the model to make the system evolvable. Our replicase system is built from reactions in the form shown in table 1. There are four events in the system which change the agents making up the metabolism at any one time. Each of these events is controlled by the sequence of codes belonging to the agents, under a stochastic timing process.

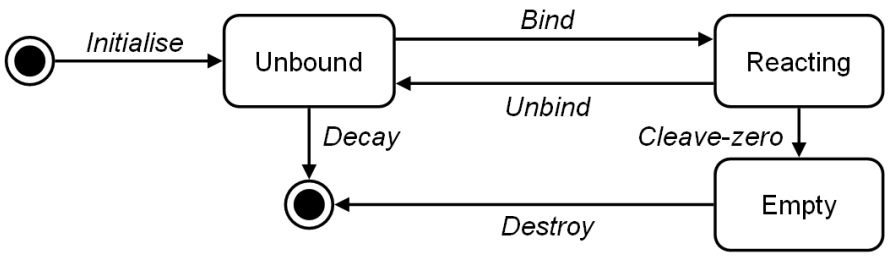


Figure 3: State chart for Agents

Binding occurs when two reactants combine to create a single product. Binding is the only bimolecular reaction permitted. Bimolecular reaction rates are governed by the concentration of the two reactants in the system (via equation 1) and a further reaction rate specified by the alignments of the molecular strings as described below. In our test model, the two agents that combine to form a bind are both instances of the replicase agent. Once agents are bound, they form a *reaction*. It is helpful to consider a reaction as a distinct computational entity. Whilst agents are in a reaction, the sequence of symbols acts as a program, making use of other structures belonging to the agent in order to carry out its function.

Production is the creation of a new agent during a reaction. Production of a new molecule is encoded as a special instruction in the sequence of one of the agents, which must be suitably positioned on the sequence in order for a new molecule to be created. The structure of the new agent is determined by the microprogram of the reaction.

Dissociation is the splitting of a reaction into its two constituent agents. This occurs after the microprogram of the reaction has terminated. The sequence of any agent may be changed by the reaction.

Decay is an essential component of our replicase model, since it allows an equilibrium state to arise where the number of replicase molecules is roughly constant. For computational simplicity, decay is spontaneous - agents are instantaneously deleted from the system with none of the degradation of structure that can occur in biological systems. Decay is stochastic, and occurs with a fixed probability of $1/65^2 = 0.237e10^{-3}$. This rate was selected for historic reasons: Version 0.1 of Stringmol used a decay rate based on length and the original replicase in [8] was 65 symbols long, and so would decay at the same rate as chosen for all molecules under the current scheme. Note also that if a molecule is bound to another molecule when it decays, then that molecule will also decay. Justification for these specifications is given in C.

It is important to note that the rates in each of the event types above are controlled by the sub-symbolic representation of the molecular agent. We specify binding rate B via an alignment of sequences. The dissociation rate is controlled by the number of time steps it takes the program specified by the sequence to execute plus a stochastic term based on availability of energy. An instruction will only be executed if energy is available. Decay is governed by the length of the string.

The relations between the objects and events in our system are illustrated in figures 2 to 4. Figure 2 is a class diagram of the two object types in our system, namely *agent* and *reaction*. An agent can participate in a reaction in three ways. Two *reagents* bind to form the reaction complex. The reaction can produce new agents in the form of *products*. Any reagents remaining when the reaction terminates are *dissociants* (there can be one or two of these).

A state chart for the agent class is shown in figure 3. The bind, unbind (i.e. dissociate) and decay events are all instantaneous. Unbound agents are essentially inert until a bind event or a decay event happens. When two agents bind, they are said to be *reacting*. During a Reaction, new agents can be produced. When the reaction ends, the partners in the reaction revert to being inert agents. An agent's string specifies its binding properties. Once in a reaction, the same string becomes the program. There are no delimiters between the binding region and the program region. A reactant can be destroyed whilst in a reaction if a cleave operation (described below) occurs at the first symbol, producing a molecule with no sequence

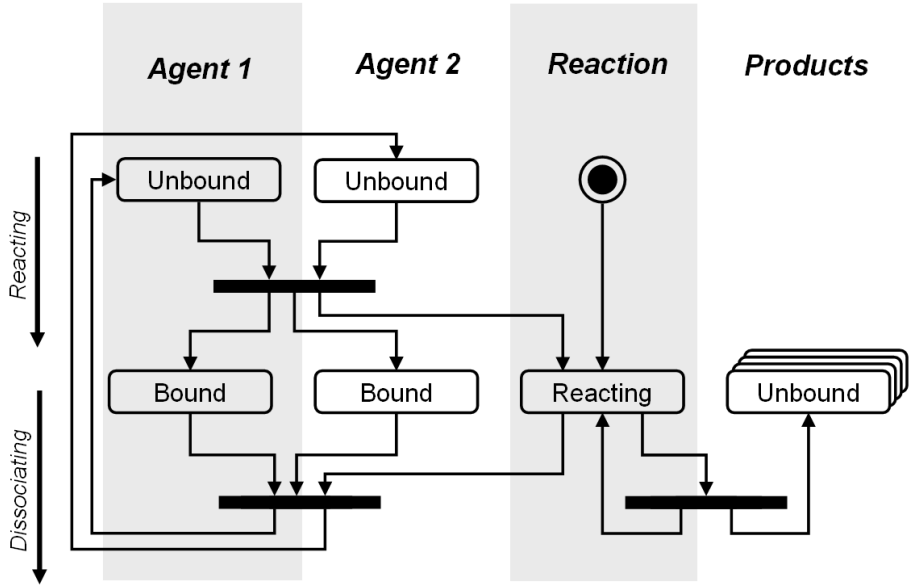


Figure 4: Activity diagram for Agents and Reactions.

(shown as "cleave-zero" in figure 3.

An activity diagram for objects in the system is shown in figure 4. It illustrates how the two agents form a reaction, and how new agents may be produced in that reaction.

5 Designing an artificial replicase

To test the concept of combining an artificial chemistry with a stack-based program specification, we set ourselves the goal of designing a self-maintaining chemical system constructed of multiple instances of a single molecular species. We call this molecule a "replicase", since it must create copies of itself to maintain the population. To do this, an instance of the molecule must bind to another instance of the molecule, and through some process create a third instance of the molecule. Following this, the original two molecules must dissociate. The molecules are also subject to a decay process, such that they effectively have a "lifetime".

Our molecular microprograms are analogous to proteins and sRNAs, where sequences of amino acids fold to form molecular machines. Our molecules have no shape or explicit dimension. The uniqueness of a molecule is encoded in its sequence of symbols - its string. We have therefore acquired the habit of calling our molecules *Stringmols*. We use program control flow as an analogy of the tertiary structure of a protein for a sequence of program symbols, since program flow is likely to change dramatically with small changes in the symbol sequence, just as the shape of a protein can change via mutation of a single amino acid. We place heavier emphasis on the process of molecular binding than do AVIDA and variants. In functioning microprograms, sequences describe one or more binding regions and when bound, a microprogram is executed from the sequence of instructions commencing at the start of the bind. Thus a binding event triggers a reaction sequence, which is encoded on the molecule and run as a program. Different programs can be encoded on the same molecule, and be triggered by binding at different sites.

The sequence of the designed replicase is shown in figure 5. There are three distinct regions. The first controls the binding probability for two agents. Region 1 of one replicase molecule binds to region 2 of another replicase molecule by a process of complementary binding as described below. The centre of the sequence is "junk". This region carries out no function, but its presence adds to the length of the sequence and is thus important in determining the decay rate of the molecule. The final region specifies the reaction

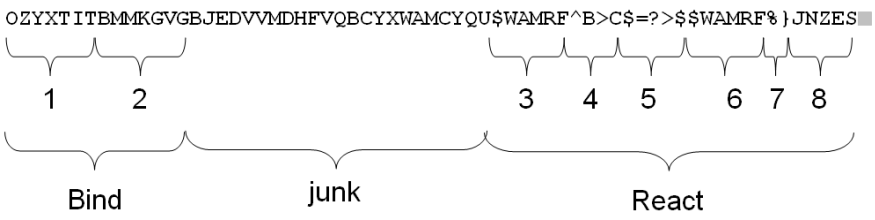


Figure 5: Sequence of an artificial replicase

that the replicase performs. It is here that the sequence can be interpreted as a program (described below). Note that the three regions of the molecule are not delimited in any way. There is thus scope to design a molecular sequence which has more than one binding site and/or program. A sequence could be interpreted as all junk if it doesn't bind. A sequence can be very short and simply bind to another molecule type to prevent it reacting with anything else. We believe that this flexibility is one of the strengths of the system.

6 Binding using inexact subsequence alignment

We require a model of biochemical binding that is simple to implement. The main idea is that when two molecules are brought together, there is a *probability* of binding - the two molecules will join together with a particular probability, or they will not join. There is no intermediate state in our abstraction, simply a before and after state. The probability of biological molecules binding is due to many things (temperature, pressure, pH), but the process that we emulate here is that the shape and composition of the two molecules is the key evolved factor in determining whether they join together. Thus, our binding scheme is concerned with some comparison of the strings of the two agents in a bind. In this section, we consider how to build a binding scheme which uses inexact sequence alignments as the basis of the probability of two agents binding. We only discuss strings here - the molecular apparatus needed to run the sequence as a program is described in the sections that follow.

Although subsequence alignments offer a method for obtaining a score for an alignment, several design decisions must be made in order to use the approach to simulate binding between molecules. An *alphabet* of symbols must be chosen. We need to ensure that two instances of the same species do not always bind strongly to each other. This can be achieved by specifying that binding occurs via a process of *complementary alignment*. We must then decide the complementary symbol of each symbol in the alphabet. An appropriate *scoring scheme* for the alignments must be available, and an appropriate function to convert the score to a *binding probability* must be developed. This section describes our design choices. In addition, section 6.4 describes the Smith-Waterman algorithm (SWa), which forms the basis of our approach. We have not changed the algorithm in any way, merely developed procedures to construct input and output functions that make it suitable for the task at hand.

6.1 Sequences of symbols

Strings are composed of sequences of symbols from a finite alphabet Σ. Symbols are of two types. Firstly, we have T, the set of twenty-six *template codes* from the uppercase alphabet:

$$T = \left\{ \begin{array}{l} \texttt{'A'}, \texttt{'B'}, \texttt{'C'}, \texttt{'D'}, \texttt{'E'}, \texttt{'F'}, \texttt{'G'}, \texttt{'H'}, \texttt{'I'}, \texttt{'J'}, \texttt{'K'}, \texttt{'L'}, \texttt{'M'}, \\ \texttt{'N'}, \texttt{'O'}, \texttt{'P'}, \texttt{'Q'}, \texttt{'R'}, \texttt{'S'}, \texttt{'T'}, \texttt{'U'}, \texttt{'V'}, \texttt{'W'}, \texttt{'X'}, \texttt{'Y'}, \texttt{'Z'} \end{array} \right\} \quad (2)$$

which are used for alignment of strings and for program flow control only. We use the complete upper case alphabet to facilitate rich sets of template sequences in our molecular structures, and to allow mnemonic

Σ	**ABC$DEF>GHIJ^KLM=NOP?QRS}TUV%WXYZ**
$C(\Sigma)$	**NOP$QRS>TUVW^XYZ=ABC?DEF}GHI%JKLM**

Table 2: The set of symbols Σ , and its complement $C(\Sigma)$.

sequences to be used whilst initialising our molecules. Secondly, we have Φ, the set of seven *function codes*, which are used for the seven different operations in our microprogramming language:

$$\Phi = \{ \text{`\$'}, \text{`>'}, \text{`\^{}'}, \text{`?'}, \text{`='}, \text{`\%'}, \text{`}\}\text{'} \} \tag{3}$$

The alphabet Σ is the union of the set of template codes T and the set of function codes Φ:

$$\Sigma = T \cup \Phi \tag{4}$$

Symbols are always written in **THIS** font in our notation.

Each molecule has a program, encoded as a string s, consisting of a sequence of codes from the alphabet. s is a member of the set of all possible strings Σ^*:

$$s \in \Sigma^*. \tag{5}$$

A symbol with index i on string s is indicated by

$$s_i, \quad i \in 0...(m-1) \tag{6}$$

where m is the length of the string.

An important property of these strings of symbol sequences is that they contains subsequences that may be aligned with other, non-contiguous subsequences

6.2 Complementary string alignments

We use an adapted form of complementary base pairing as part of our alignment strategy. The complement of a string s is the string $\mathrm{C}(s)$. The complement of all symbols in Σ is shown in table 2. Our (untested) intuition is that template symbol complements could be chosen arbitrarily, but that functional symbol complements offer a potential hazard — they may force long functional regions to act as templates, and thus suppress the generation of alternative binding schemes. In these early stages of designing this chemistry, we have specified the complement of a template symbol to be 13 letters downstream in the English alphabet and that functional symbols act as complements of themselves. Note also that the *order* of the symbols is important, as it is used to calculate mismatch scores in the alignment process - the nearer a symbol is in the list, the less severe the mismatch.

The process of alignment is shown in figure 6. The sequence '**OZYXTIT**' from region 1 of string 2 is bound to the sequence '**BMMKGVG**' from region 2 of string 1, since the complement of region 2 forms an alignment with region 1. The complementary nature of the alignment is necessary to prevent copies of strings binding to each other perfectly, and to allow more than one instance of a sequence on a string to align to a single complementary sequence. As we shall see, this factor also allows subroutines ("goto" statements) to be developed, which the microprograms can exploit whilst functionality is evolving.

6.3 A substitution matrix for microcodes

Alignments are considered by comparing strings on a symbol-by-symbol basis. A *score* must be available for a match, a mismatch, or an indel (an insertion or deletion of a symbol). When attempting to find the best alignment between string s with index i and length m, and string s' with index j and length n, it is

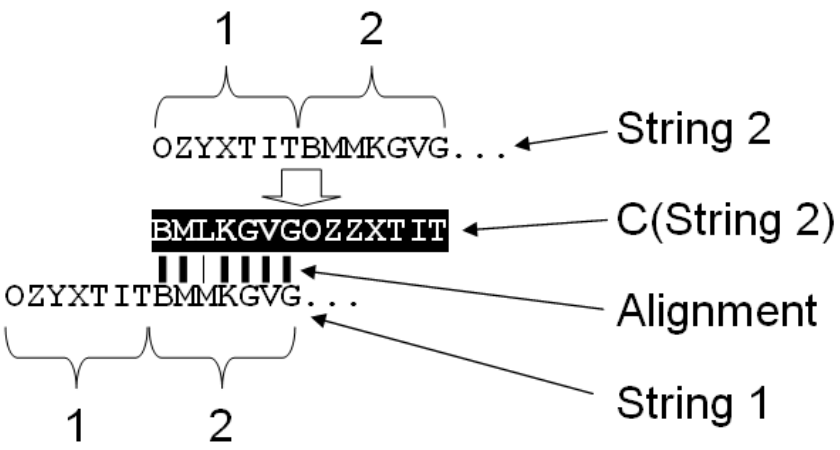


Figure 6: Complementary matching to form a bind. The sequence '**OZXYTIT**' has a complement (shown in white lettering) which aligns with '**BMMKGVG**' . The alignment is not perfect as the third letter in the sequence forms a mismatch.

desirable to have a scoring system that reflects the chance of a mutation (or indel) from the symbol at s_i to the symbol at s'_j.

In studies of biological sequences, the chance of mutation is estimated by reference to large numbers of studied sequence phylogenies. When comparing two sequences of symbols s and s', three cases need to be assigned a score: a match from one symbol to another; a mutation from one symbol to another; and a gap in the alignment caused by the insertion of a symbol on one string or a deletion on the other (an *indel*). These values are stored in a substitution matrix, ω. This matrix is of size $(n(\Sigma) + 1) \times n(\Sigma)$, where $n(\Sigma)$ is the number of symbols in the alphabet. The extra row is used to store the indel score for a particular symbol and has index $-$. The function $\overrightarrow{\Sigma}(s_i)$ returns the index on ω of the symbol at s_i.

For sequences from biological systems the evidence for mutation rates that have occurred historically is used to give a score to the differences in the sequences under consideration. Even though our model has no mutation at present, we still require a substitution matrix to obtain an alignment of strings via SWa. In our application, we must generate substitution values from scratch in the absence of a historical record of mutations.

Unconstrained by the physical world, we are free to define the substitutions between symbols in any way we choose. As a first step, we have specified that codes mutate forward or backward through a sequential loop. The functional symbols are intermingled with the template symbols throughout the loop to allow functional changes for any symbol to be always only a few mutations away and reduce any clustering of functional codes to form "functional hotspots" in mutation sequences. Substitution mutations can be forward or backward in the sequence. Thus the symbol '**D**' can mutate to one or other of its neighbours: '**$**' or '**E**' , and at the start of the loop the symbol '**A**' can mutate to '**B**' or (going to the end of the sequence) '**Z**' . (See table 2 for the full sequence.) A change from '**C**' to '**J**' would require at least seven mutations (if there happen to be no anti-clockwise mutations), and '**L**' to '**S**' would require at least eight mutations.

It is straightforward to implement this mutation scheme in an evolving system. Fully random mutations of fixed probability between all codes would create different substitution values. It is difficult to know *a priori* what weight a scheme such as this will have on the efficacy of an evolutionary algorithm. The potential benefits are that codes with closely-related properties can be given "cheap" substitution scores, and thus allow smoother exploration of the gene-space. In this current contribution, we can only speculate about the effects of this approach.

	A	**B**	**C**	**$**	**D**	**E**	**F**	**>**	**G**	**H**
A	1.00	−0.12	−0.25	−0.38	−0.50	−0.62	−0.75	−0.88	−1.00	−1.12
B	−0.12	1.00	−0.12	−0.25	−0.38	−0.50	−0.62	−0.75	−0.88	−1.00
C	−0.25	−0.12	1.00	−0.12	−0.25	−0.38	−0.50	−0.62	−0.75	−0.88
$	−0.38	−0.25	−0.12	0.50	−0.12	−0.25	−0.38	−0.50	−0.62	−0.75
D	−0.50	−0.38	−0.25	−0.12	1.00	−0.12	−0.25	−0.38	−0.50	−0.62
E	−0.62	−0.50	−0.38	−0.25	−0.12	1.00	−0.12	−0.25	−0.38	−0.50
F	−0.75	−0.62	−0.50	−0.38	−0.25	−0.12	1.00	−0.12	−0.25	−0.38
>	−0.88	−0.75	−0.62	−0.50	−0.38	−0.25	−0.12	0.50	−0.12	−0.25
G	−1.00	−0.88	−0.75	−0.62	−0.50	−0.38	−0.25	−0.12	1.00	−0.12
H	−1.12	−1.00	−0.88	−0.75	−0.62	−0.50	−0.38	−0.25	−0.12	1.00
-	−1.33	−1.33	−1.33	−1.33	−1.33	−1.33	−1.33	−1.33	−1.33	−1.33

Table 3: Sample values of the substitution matrix ω. Indel scores are presented on the bottom row of the table

Although this strategy allows us to give *relative* values of substitution scores, we need to fix the values in absolute terms in order to create the substitution matrix. Smith and Waterman [9] state that for a four-letter alphabet, a substitution weight of $-1/3$, a match weight of 1 and an indel weight of $-4/3$ gives an average score of zero for randomly-generated strings. Shpaer [15] reports a substitution matrix for amino acids. This gives similar average substitution weights if the values are normalised for the average match score.

Our substitution matrix ω is filled as follows. Let $a = \overrightarrow{\Sigma}(s_i)$ and $b = \overrightarrow{\Sigma}(s'_j)$. The diagonal values of the matrix, representing the score if symbol a is found at s_i and s'_j, are set to 1.0 for template codes, and 0.5 for function codes:

$$\omega(a,a) = \begin{cases} 1.0 & \text{if } a \in T \\ 0.5 & \text{if } a \in \Phi \end{cases} \tag{7}$$

This places appropriate emphasis on the binding and executional function of the two symbol types. These scores occupy the leading diagonal of ω as shown in table 3. Scores for mismatches are based on the distance between symbols in the alphabet, and are calculated via the *substitution equation*:

$$\omega(a,b) = \frac{-\min(|a-b|, N-|a-b|)}{\sum_{k=1}^{N} \min(k, N-k)/N} \qquad \text{if } (a \neq b) \tag{8}$$

where the size of the alphabet $N = n(\Sigma)$. This sets the average score for substitution to -1. This average penalty is more severe than Smith and Waterman recommend, and has the effect that more alignments will score zero (see below). We have adopted this strategy because our application currently utilises alignments that are significantly shorter than the biological sequence alignments that SWa was designed for. A more severe penalty on mismatches helps to ensure that accidental alignments occur at an appropriate rate in our model. Note also that substitution scores depend entirely on the sequence in which symbols from Σ are placed in ω. Scores for indels are fixed to a constant:

$$\omega(-,a) = -4/3 \tag{9}$$

These indel values are shown on the bottom row of table 3. The penalty for substitution gets more severe with distance from the leading diagonal, indicating that a *sequence* of mutations would be required for distant symbols to be substituted for one another.

6.4 Calculating the alignment score using the Smith-Waterman algorithm

With our substitution matrix to hand, we can calculate the highest-scoring alignment for any pair of strings formed from Σ using the Smith-Waterman (SWa) algorithm. For two sequences s and s' of length m and n respectively, the algorithm populates a matrix H of size $m+1$ by $n+1$ with the similarity scores between all symbols in two sequences s and $C(s')$. Each cell in the matrix H is initialised to zero. The first row and the first column in H act as indicators for the start of the strings. The presence of these extra cells allows processing to be done in a uniform manner. The algorithm starts at cell $H(i=1, j=1)$ (assuming zero-indexing), and proceeds row by row until cell $H(i=m, j=n)$ is reached. The value of each cell in this submatrix is [16]:

$$H(i,j) = \max \begin{cases} 0 & \text{No alignment} \\ H(i-1,j-1) + \omega(a,b) & \text{Match/Mismatch} \\ H(i-1,j) + \omega(-,a) & \text{Deletion} \\ H(i,j-1) + \omega(-,b) & \text{Insertion} \end{cases} \tag{10}$$

Where $w(x,y)$ returns the appropriate value from the substitution matrix at x, y and $w(-,x)$ indicates the row containing the indel value for symbol x. Scores for sequences of pure matches and mismatches run parallel to the leading diagonal of H, whereas insertions and deletions cause a horizontal or vertical shift in the path of the maximal score. In addition to H, a *trace matrix* T is produced, which indicates whether the value in each cell is the result of a match/mismatch, insertion or deletion. B illustrates this process with an example.

Once processed, the maximum local alignment $\phi_{S,S'}$ can be found by detecting the highest value $H(i_\beta, j_\beta)$ in the matrix H, and then tracing the alignment back to its origin using T. The origin of the alignment is the cell $H(i_\alpha, j_\alpha)$, in which all upper-left neighbours are zero-valued is obtained. The alignment site on s runs from s_{i_α} to s_{i_β} and the alignment on s' runs from s'_{j_α} to s'_{j_β}.

Let the index on s and s' of the zero-scoring cell in H which marks the start of the alignment be s_α and s'_α and the index of the highest-scoring cell in H which marks the end of the alignment be s_β and s'_β. The length λ of the bind is the minimum length of the alignment on the two strings:

$$\lambda = \min(s_\beta - s_\alpha, s'_\beta - s'_\alpha) \tag{11}$$

Note that the minimum alignment length can never be less than the alignment score. Alternative schemes might make use of the maximum length alignment.

Given the ingredients above, we have implemented a function ϕ, which takes as arguments two strings, s and s' and a substitution matrix ω. The function finds the Smith-Waterman alignment between s and $\mathrm{C}(s')$, and returns the score σ and length λ of the alignment:

$$\sigma_{s,s'}, \lambda_{s,s'} = \phi(s, s', \omega) \tag{12}$$

6.5 From alignment scores to bind probabilities

The Smith-Waterman algorithm was designed to give a score to subsequence alignments for the analysis of phylogeny. It was not intended for use as a metric for binding. In order to use SWa for binding, we must define a function which maps the alignment score to a probability of bind success. Note also that bind *success* is distinct from bind *strength*. The strength of a bind, once formed is uniform and absolute. The bind can only be broken by terminating the program that is run when the bind is formed.

In order to calculate the alignment probability for any two strings s and s', we first obtain $\phi(s, s', \omega)$, and use the score σ and the length λ to define the binding site and derive a probability of binding, $\mathrm{P}(\phi(s, s', \omega))$. Longer subsequences tend to score more highly. We can take these values and derive a binding probability via reference to the maximum possible score for a match of a given length.

For an alignment of score σ, we define the binding probability P as:

$$\mathrm{P}(\phi(s, s', \omega)) = \begin{cases} 0 & \text{if } \lambda <= 3 \\ \\ \min(\sigma, \lambda - m)/(\lambda - m) & \text{otherwise} \end{cases} \tag{13}$$

where m is the penalty for a single-point mismatch in the Smith-Waterman alignment matrix. This equation gives us the range of binding strengths we need to build metabolic networks of computational entities. If a particularly weak bind is sought, then a longer subsequence alignment with sufficient mismatches can deliver that weak bind.

We have now shown how to obtain a bind from a string alignment. When two agents bind successfully, they form a reaction. Before we describe how reactions are run as microprograms as specified by the agent sequences, we must detail the apparatus that is needed to carry out the reaction.

7 Molecular structure

In order to use the string sequences to drive a range of intermolecular reactions and thus develop new molecules with their own sequences, we require some means of using the symbol sequence to specify manipulation of the strings. In this section, we describe the structures that we use to allow molecules to run string microprograms. Recall our biochemical analogy: two adjacent molecules binding to each other, initiating a reaction that creates products. The characteristics of all of these events are encoded on the string of the molecular analogues. In addition, each molecule has an architecture consisting of a set of pointers and flags that govern the execution of the program. The key concept here is that a bound pair of molecules forms a complete computational entity. The sequences of both molecules are used in the program. Pointers exist in pairs, with only one of the pair ever "active" at any one time. The "active" pointers belonging to a single molecule in the bound pair control program flow and read/write operations, and are similar to the pointers used in the AVIDA system [11]. A record of which pointer is active in each pair is stored in "activator" flags. Finally, the state of the molecule is recorded in a "status" flag.

In this section, we denote a molecule as $\mathcal{M}$, and use the operator ':' to denote structures belonging to the molecule. For example the string s^1 of a molecule $\mathcal{M}^B$ is referred to as $\mathcal{M}^B : s^1$. We use the notation $\perp$ to indicate where structures are undefined. Each molecule $\mathcal{M}$ has access to a string s^1, a string from a partner molecule s^2, two sets of pointers p^1 and p^2, a set of activators a, and a status flag δ:

$$\mathcal{M} \begin{cases} s^1, s^2 & \in \Sigma^* \cup \perp \\ p^1 & \in \mathbb{N}^4 \\ p^2 & \in \mathbb{N}^4 \cup \perp \\ a & \in \{1, 2\}^4 \\ \delta & \in \{\mathrm{UNB}; \mathrm{ACT}; \mathrm{PAS}\} \end{cases} \tag{14}$$

7.1 Molecular states

Our computational molecules exist in one of three states δ: unbound (UNB); active (ACT); passive (PAS):

$$\delta = \{\mathrm{UNB}; \mathrm{ACT}; \mathrm{PAS}\} \tag{15}$$

These states govern the availability of the other structures that comprise the molecule as a reaction proceeds.

When two molecules bind, one molecule takes an active state and one takes a passive state. We adopt the convention that the active molecule has index 1 and the passive molecule has index 2, reflecting the "grammatical person" in the English language (the first person is "I" and the second person is "you"). The molecule in the active state (δ = ACT) runs the reaction initiated by the bind.

7.2 Molecular strings

The *sequence* s^1 of a molecule defines the molecule's binding and program. The reaction between two molecules is encoded on the strings of the two molecules. We use the notation $\mathcal{M} : s^2$ to reference the string of the partner of a bound molecule. The definition of a sequence is shown in equation 5.

To illustrate this concept, consider two bound molecules, with sequences s^A and s^B. The active molecule $\mathcal{M}^A$ has $\mathcal{M}^A : s^1 = s^A$; $\mathcal{M}^A : s^2 = s^B$; $\mathcal{M}^A : \delta = \mathrm{ACT}$. The passive molecule $\mathcal{M}^B$ has $\mathcal{M}^B : s^1 = s^B$; $\mathcal{M}^B : s^2 = s^A$; $\mathcal{M}^B : \delta = \mathrm{PAS}$. By contrast, an unbound molecule $\mathcal{M}^U$ with sequence s^U has $\mathcal{M}^U : s^1 = s^U$; $\mathcal{M} : s^2 = \perp$; $\mathcal{M}^U : \delta = \mathrm{UNB}$.

7.3 Pointers

Programs are executed when molecules bind, via manipulation of pointers and the symbols that they point to. There are four pointer types: instruction $p_{\mathbf{I}}$; flow $p_{\mathbf{F}}$; read $p_{\mathbf{R}}$; write $p_{\mathbf{W}}$. Pointers take the value of the index of the symbol on a molecule's string s to which they are pointing. The set of four pointers is:

$$p = \langle p_{\mathbf{I}}, p_{\mathbf{F}}, p_{\mathbf{R}}, p_{\mathbf{W}} \rangle \in \mathbb{N}^4 \tag{16}$$

Each molecule has two sets of the four pointers. One set, p^1 is used to interact with the string of the molecule itself, the other, p^2, is used to interact with the string of a partner molecule in the reaction. Only one instance of each pointer type is ever available for interaction at any one time. Pointers are only set to a defined value whilst the molecule is active (i.e. $\mathcal{M} : \delta = \mathrm{ACT}$).

7.4 Activators

Activators specify which of the two instances of each pointer type is currently active. Activators take values:

$$a = \langle a_{\mathbf{I}}, a_{\mathbf{F}}, a_{\mathbf{R}}, a_{\mathbf{W}} \rangle \in \{1, 2\}^4 \tag{17}$$

Each member of $\mathcal{M} : a$ takes one of two values 1 and 2, indicating which pointer set $\mathcal{M}{:}p^1$ or $\mathcal{M}{:}p^2$, and thus which molecule's string $\mathcal{M}{:}s^1$ or $\mathcal{M}{:}s^2$, that a particular pointer type is currently acting upon. All activators are set to 1 when a molecule's state becomes active, so placing all the pointer types on the active molecule.

7.4.1 Binding molecules

When molecules are created, their state is unbound: $\mathcal{M} : \delta = \mathrm{UNB}$. They are not in the process of reacting, and so no pointer manipulation is taking place. Precisely two molecules are involved in a binding event. Within this bind, one of the molecules is considered to be executing its program, much as a biological enzyme can be thought to be acting upon a substrate. By convention, we specify that molecule $\mathcal{M}^1$ is the *active molecule* the molecule which is executing the program, with state $\mathcal{M}^1 : \delta = \mathrm{ACT}$ and molecule $\mathcal{M}^2$ is the *passive molecule* so $\mathcal{M}^2 : \delta = \mathrm{PAS}$. When two molecules $\mathcal{M}^A$ and $\mathcal{M}^B$ are brought together, we must first test whether they bind, and if they do bind, we must determine which becomes the active molecule $\mathcal{M}^1$ and which becomes the passive molecule $\mathcal{M}^2$.

To determine whether the molecules bind, an alignment test is carried out to obtain the binding probability $\mathrm{P}(B) = \mathrm{P}\left(\phi(\mathcal{M}^A : s^1, \mathcal{M}^B : s^1, \omega)\right)$ and arrange the apparatus of each molecule accordingly. If a bind is successful, a handshaking process is required to determine at which symbol the execution of the reaction starts. For two sequences s and s' the executing sequence s^1 is that for which the start of the binding site s_α is furthest from the start of the sequence. To determine whether a bind has occurred, $\mathrm{P}(B)$

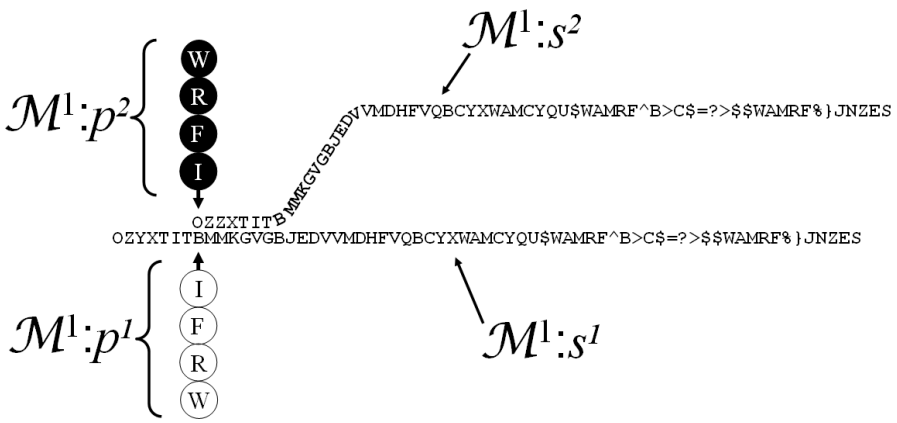


Figure 7: Initial state of a reaction between two replicase agents. Since $\mathcal{M}^1 : a := \{1\}^4$, all active pointers (shown as black text in white circles) are on $\mathcal{M}^1 : s^1$.

is compared with a random number r drawn from the range $[0, 1)$. The state of the two molecules changes as follows:

$$
\begin{array}{l}
\text{if}(r < \mathrm{P}\,(B)) \\
\quad \text{if}(\mathcal{M}^A : s^1_\alpha <= \mathcal{M}^B : s^1_\alpha) \\
\quad\quad \mathcal{M}^1 := \mathcal{M}^A; \mathcal{M}^2 := \mathcal{M}^B \\
\quad \text{else} \\
\quad\quad \mathcal{M}^1 := \mathcal{M}^B; \mathcal{M}^2 := \mathcal{M}^A \\
\quad \mathcal{M}^1 : \delta := \mathrm{ACT}; \mathcal{M}^2 : \delta := \mathrm{PAS} \\
\quad \mathcal{M}^1 : s^2 := \mathcal{M}^2 : s^1 \\
\quad \mathcal{M}^1 : p^1 := (\mathcal{M}^1 : s^1_\alpha)^4 \\
\quad \mathcal{M}^1 : p^2 := (\mathcal{M}^1 : s^2_\alpha)^4 \\
\quad \mathcal{M}^1 : a := (1)^4
\end{array}
\tag{18}
$$

We would like the properties of the bind to act as determining factors for making this choice. Although there are many ways to decide which molecule is most appropriate to name as the executing sequence, we have gone for simplicity here, and use the distance of the start of bind to the start of each sequence to determine which sequence executes.

In our replicase example, this strategy makes scanning of the entire sequence of the passive molecule simpler if the complementary site is placed at the beginning of the sequence, since there is then no need to reposition the read pointer after binding. This is illustrated in figure 7. The active molecule's string s^1 is placed below the passive molecule's string s^2 in the figure. The active pointers are all placed at the start of the bind on s^1 (so a^1 = 1), and are shown in white with black text. The inactive pointers are all at the start of the bind on S^2, which happens to be the beginning of the string.

8 Overview of the Microcode library

We have implemented a limited set of *microcodes* - special symbols in the alphabet that allow us to perform computational analogues of molecular reactions. Table 4 provides a summary of these microcodes, which together manipulate the pointers and control the execution pathways of the molecular microprograms. We have carefully designed our codes such that they only have influence on the bound pair of strings. These instructions manipulate the positions of pointers, and so control the execution pathways of the molecular microprograms. In the sections below, we indicate the symbols and names for our codes in bold courier font.

Code(s)	Name	Default pointer	Description
A to **Z**	**n-op**		inactive template code and instruction modifier
$	**search**	***F**	shift ***F** to a matching template
>	**move**	***I**	shift pointer to the flow pointer
^	**toggle**	***I**	switch pointer to molecular bind partner
?	**if**	***I**	conditional single or double increment of instruction pointer
=	**copy**		insert at ***W** a copy of the symbol at ***R**
%	**cleave**		split a sequence and generate a new molecule
}	**end**		terminate execution and break the bond between the molecules

Table 4: Symbols and actions used in the current implementation of molecular microprograms.

Example	Modifiers	Associated Template
?$$	No modifiers	No associated template
?A$	The subject pointer becomes ***I**, the instruction pointer.	No associated template
?B$	The subject pointer becomes ***R**, the read pointer	No associated template
?C$	The subject pointer becomes ***W**, the write pointer	No associated template
?BNZRY$	No modifiers.	The associated template is **BNZRY**

Table 5: Examples of modifiers and associated templates

8.1 Modifiers and associated templates

Microcodes work by shifting the position of pointers and manipulating symbols and structures at the pointer positions. We would like to limit the number of different microcodes in the system, but make them as flexible as possible by allowing them to operate on different pointers in certain circumstances. This is implemented by specifying *modifiers* to operational codes. On execution of an operational code, the microcode immediately downstream is checked. If it is a modifying code, then the pointer to which the executing code applies is substituted according to the modification. Instructions that make use of pointers always have a *default pointer*. This is the pointer that is used if a modifier is not present.

In addition to manipulating the pointer that a code operates on, it is also necessary to have some mechanism that allows pointers to make a comparison between sequences on strings. The general mechanism for this is to define a sequence immediately downstream of the microcode that is the complement of a subsequence elsewhere on the string. A series of template codes downstream of a functional code is the *associated template* of the functional code. The uses of the associated template are described below.

Table 5 shows examples of modifiers and associated templates. The *subject pointer* is the pointer that is subject to manipulation. There is always a default subject pointer, but the subject pointer can be changed by reference to modifiers.

Note that only three modifiers are used: **A**, **B** and **C**, so only $3/34$ instructions are modifiers, meaning that on average, the default pointer would be used 85% of the time ($1 - (3/34)$). This is probably a conservative bias, and we may want to increase the number of modifiers in subsequent work - possibly using all 26 template codes as modifiers.

8.2 Notation

In the following descriptions it is convenient to simplify some of the notation above. Since the active molecule in the reaction has all the apparatus required to run that reaction, we drop the $\mathcal{M}$ prefix in this section. Most of our microcode operations refer only to pointers which are currently active. The activators a indicate which pointer is currently active. Since only active pointers are used for most operations, it is convenient to drop the a where only active pointers are used. For example, $s_{\mathbf{F}}^{a_{\mathbf{F}}}$ indicates the symbol with index $\mathbf{F}^{a_{\mathbf{F}}}$ on the string that the active flow pointer currently references. Where this is clear, we drop the $a_{\mathbf{F}}$ and use the notation $s_{\mathbf{F}}$ to make the notation easier to read.

9 Auxiliary functions

These functions act as the building-blocks for the functional microcodes. We felt that they were too simple for their function to be represented as a unique code since there would be too many codes required for a novel function to emerge.

9.1 len

This function returns the length of a string. Thus for a string of length 12, $\mathrm{len}(s) = 12$. If the function is applied to a pointer, the function returns the length of any associated template, which is the number of template codes downstream of the pointer until a function code or the end of the string is encountered. Thus if a pointer p indexes the start of the sequence **`$JAMES%^`**, $\mathrm{len}(p) = 5$.

9.2 rand

The $\mathrm{rand}\,()$ function returns a pseudo-random number in the range $[0, 1)$.

9.3 swap

Each molecule possesses two instances of each pointer type. The pointer that the functions act on is specified by the activators $\mathcal{M}^1 : a$. The swap function acts on $1, 2 \rightarrow 2, 1$. Thus in the case of the instruction pointer:

$$\begin{array}{l}\mathrm{swap}(x):\\ \quad \mathrm{return}(x == 1?2:1)\end{array} \tag{19}$$

9.4 modify pointer

Some microcodes are subject to modifiers, which change the pointer that the microcodes act upon. The modifier function acts on the tuple $< p_{\mathbf{I}}, p_{\mathbf{F}}, p_{\mathbf{R}}, p_{\mathbf{W}} >$

$$\begin{array}{l}\mathrm{mp}\,(p_{\mathbf{X}}^{a}):\\ \quad \mathrm{switch}(s_{\mathbf{X}+1})\\ \qquad \text{case '}\mathbf{A}\text{': return } p_{\mathbf{I}}\\ \qquad \text{case '}\mathbf{B}\text{': return } p_{\mathbf{R}}\\ \qquad \text{case '}\mathbf{C}\text{': return } p_{\mathbf{W}}\\ \qquad \text{default: return } p_{\mathbf{X}}\end{array} \tag{20}$$

where $\mathbf{X} \in \langle \mathbf{I}, \mathbf{F}, \mathbf{R}, \mathbf{W} \rangle$.

9.5 template match position

Our microprograms make use of the alignment of template sequences to execute 'goto'-like control of program execution. Since templates are used to position pointers, we can use inexact template matches to introduce stochasticity into execution of the code of the molecule. Unlike the scenario of binding, the length of the best alignment is known in advance — it is the length of the template sequence that follows the instruction that is immediately upstream of it. For example, the sequence **`$JAMES%^`** has length $\lambda = 5$, the length of the sequence of template codes after the **`$`**. Rather than using the *maximum* length of the matched sequences, we always use the length of the template as the basis for our probability calculation, as in equation 13.

$$\begin{array}{l} \mathrm{apos}\left(p^a_{\mathbf{pF}}\right): \\ \quad \sigma, \lambda = \phi(s^{a_{\mathbf{F}}}, s_{\mathbf{F}} \ldots s_{\mathbf{F}+\mathrm{len}(\mathbf{F})}, \omega) \\ \quad \text{if } \sigma > \mathrm{rand}\,() \\ \quad\quad \text{return } s^{a_{\mathbf{F}}}_{\beta} \\ \quad \text{else} \\ \quad\quad \text{return } p_{\mathbf{F}} \end{array} \tag{21}$$

where $s^{a_{\mathbf{F}}}_{\beta}$ is the end of the alignment produced by ϕ. We can thus use the alignment score to derive probabilities of execution path selection for active instructions. A slight mismatch in the sequences that delimit an otherwise infinite loop would cause the loop to be exited with a certain probability each cycle. Such a loop could act as a stochastic timer for example, with the chance of escape from the loop being inversely proportional to the strength of the alignment.

9.6 template match score

The score of an alignment between an associated template and a string can also be used by microcodes. The function $\mathrm{asco}\,(p^a)$ works in a similar manner to the way to $\mathrm{apos}\,(p^a)$, by aligning an associated template with a string. The difference is that this function returns the probability of the alignment rather than the position:

$$\begin{array}{l} \mathrm{asco}\left(p^a_{\mathbf{pF}}\right): \\ \quad \text{return } P\left(\phi(s^{a_{\mathbf{F}}}, s_{\mathbf{F}} \ldots s_{\mathbf{F}+\mathrm{len}(\mathbf{F})}, \omega)\right) \end{array} \tag{22}$$

10 Microcodes

10.1 `A` to `Z`: Template codes

These are represented by characters from the uppercase alphabet, and are used to form the templates for the matching process detailed above. When a template code is executed by the instruction pointer, its only effect is to increment the instruction pointer:

$$p^a_{\mathbf{I}} := p^a_{\mathbf{I}} + 1 \tag{23}$$

10.2 `^`: `toggle`

Toggle changes which instance of the pair of pointers of a particular type is currently active. This has the effect of switching which of the two strings the pointer type is currently active upon:

$$\begin{array}{lcl} a_{\mathrm{mp}(p^a_{\mathbf{F}})} & := & \mathrm{swap}\left(a_{\mathrm{mp}(p^a_{\mathbf{F}})}\right) \\ p^a_{\mathbf{I}} & := & p^a_{\mathbf{I}} + 1 \end{array} \tag{24}$$

In our replicase example, toggle is used to shift the active read pointer onto s_2 so that s_2 will be copied (see **^B** in region 4 of figure 5).

10.3 $: search

This operator moves $p^a_{\mathbf{F}}$ to a position corresponding to a template match $\mathrm{apos}\,(p^a_{\mathbf{F}})$. Note that $p^a_{\mathbf{F}}$ can be positioned on the active or the passive string, depending on the value of $a_{\mathbf{F}}$. Only the string to which $p^a_{\mathbf{F}}$ points at the start of execution is searched. If a match exists, **search** then aligns the flow pointer with the end of that match. $p^a_{\mathbf{I}}$ is always incremented. Where identical scores are encountered, the first such score along the string is the winning match. If no template exists, $p^a_{\mathbf{F}}$ is positioned to the next instruction after the **search**. This function is not subjected to modifiers.

$$p^a_{\mathbf{F}} \;:=\; \begin{cases} p^a_{\mathbf{I}} + 1 & \text{if there is no associated template for } p_{\mathbf{I}} \\ \mathrm{apos}\,(p^a_{\mathbf{F}}) & \text{if } r < \mathrm{P}\,(\mathrm{apos}\,(p^a_{\mathbf{F}})) \\ p^a_{\mathbf{F}} & \text{otherwise} \end{cases} \tag{25}$$

$$p^a_{\mathbf{I}} \;:=\; p^a_{\mathbf{I}} + 1$$

The search function is used in regions 3 and 6 of of our replicase molecule in figure 5 to move the flow pointer to the end of region 8.

10.4 >: move

This function shifts $p^a_{\mathbf{I}}$ to the position of $p^a_{\mathbf{F}}$, unless $p^a_{\mathbf{I}}$ is already in that position, in which case $p^a_{\mathbf{I}}$ is simply incremented. Note that if $p^a_{\mathbf{I}}$ is pointing to a different string to $p^a_{\mathbf{F}}$, then it must still be moved to the position of $p^a_{\mathbf{F}}$ (an implicit use of **toggle**). This instruction is commonly used to construct execution loops. **move** can move any of the pointers if the appropriate modifier is encountered:

$$\mathrm{mp}\,(p^a_{\mathbf{I}}) \;:=\; \begin{cases} p^a_{\mathbf{F}} & \text{if } \mathrm{mp}\,(p^a_{\mathbf{I}}) \neq p^a_{\mathbf{I}} \\ p^a_{\mathbf{I}} + 1 & \text{otherwise} \end{cases} \tag{26}$$

$$p^a_{\mathbf{I}} \;:=\; p^a_{\mathbf{I}} + 1 \qquad \text{if } \mathrm{mp}\,(p^a_{\mathbf{I}}) \neq p^a_{\mathbf{I}} \text{ AND } p^a_{\mathbf{I}} = p^a_{\mathbf{F}}$$

10.5 ?: if

This function is a way of conditionally moving $p_{\mathbf{I}}$, depending on whether some test returns TRUE or FALSE. When TRUE or FALSE is returned, $p_{\mathbf{I}}$ is incremented twice or once respectively. It can be used to check whether some iterative task has been completed, or it can be used as a stochastic timer by using an inexact match score compared with a random number.

if has a number of different functionalities, depending on the presence of modifiers or an associated template. If there is no associated template, it can be used to check if $p_{\mathbf{R}}$ is pointing at a valid string symbol, or if it has incremented beyond the last character on a string.

If an associated template is present, **if** uses a template match to carry out the Boolean test. Using equation 13, $\mathrm{P}(\phi(a, b, \omega))$ is calculated an compared with a random number r between 0 and 1. The score

for the match is tested against a random number, and **`if`** returns:

$$p_{\mathbf{I}}^{a} := \begin{cases} p_{\mathbf{I}}^{a} + \mathrm{len}(p_{\mathbf{I}}^{a}) + 2 & \text{if } r <= \mathrm{P}\left(\mathrm{asco}\left(p_{\mathbf{R}}^{a}\right)\right) \\ p_{\mathbf{I}}^{a} + \mathrm{len}(p_{\mathbf{I}}^{a}) + 1 & \text{if } r > \mathrm{P}\left(\mathrm{asco}\left(p_{\mathbf{R}}^{a}\right)\right) \\ p_{\mathbf{I}}^{a} + 2 & \text{if there is no associated template for } p_{\mathbf{I}}^{a} \text{ AND } p_{\mathbf{R}}^{a} > \mathrm{len}(s^{a_{\mathbf{R}}}) \\ p_{\mathbf{I}}^{a} + 1 & \text{otherwise} \end{cases} \tag{27}$$

The function **`if`** can be used to escape from loops if a **`move`** command is one instruction downstream of the microcode. Figure 8 illustrates this. The **`if`** has the associated template **`JNZES`**. If $p_{\mathbf{R}}$ is not pointing at **`WAMRF`** (the complement of **`JNZES`**), then $p_{\mathbf{I}}$ is incremented to the **`>`** command, and then to $p_{\mathbf{F}}$ on the subsequent execution step. If $p_{\mathbf{F}}$ is upstream of **`?`**, the program is in a loop. The loop is escaped from when $p_{\mathbf{R}}$ is moved (via a **`=`** operator) to point to **`WAMRF`**. Only then does execution escape the loop because $p_{\mathbf{I}}$ is incremented two instructions downstream of the associated template. Region 5 in figure 5 carries out this sort of functionality to iteratively copy the partner molecule.

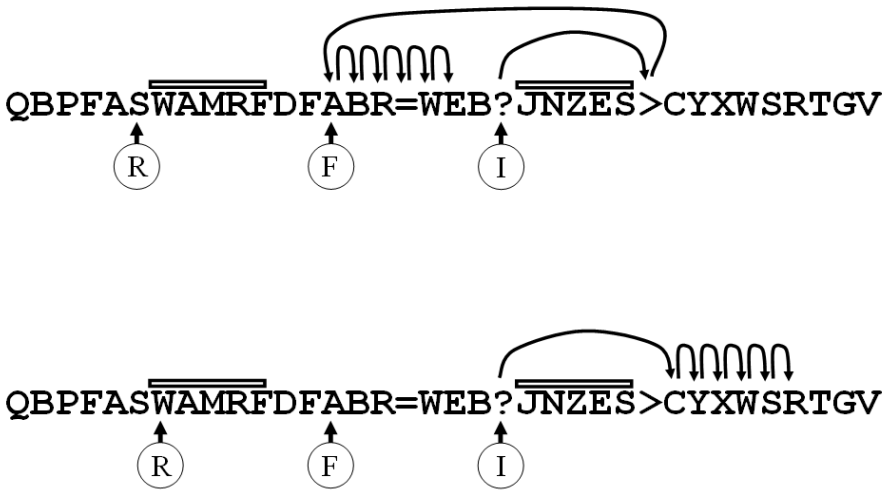


Figure 8: A program loop constructed using an **`if`** microcode. The instruction pointer follows a loop (shown top) while $p_{\mathbf{R}}^{a}$ does not match the associated template of $p_{\mathbf{I}}^{a}$. The loop terminates (shown bottom) when $p_{\mathbf{R}}^{a}$ points at a complementary match to the associated template of $p_{\mathbf{I}}^{a}$.

10.6 `=:` `copy`

This function inserts a copy of the code pointed to by $p_{\mathbf{R}}$ at $p_{\mathbf{W}}$. **`copy`** is commonly used to append the products of some reaction to the end of the executing string. If $p_{\mathbf{R}}$ is not pointing at a valid code, it has been incremented past the end of a string. If this is the case, no copy is carried out and the instruction pointer is simply incremented.

This instruction is the means by which strings increase in length. There exists a risk that programs can get trapped in loops such that they create infinitely long strings. To counter this, we specify a maximum string length Ω (Ω = 512 in [8]). If either $p_{\mathbf{R}}$ or $p_{\mathbf{W}}$ go beyond this limit, **`copy`** acts as an **`end`** instruction.

There is scope to use modifiers to change which pointer **copy** reads from. This is not currently implemented.

$$
\begin{array}{l}
\text{if } p^a_{\mathbf{R}} < \Omega \text{ AND } p^a_{\mathbf{W}} < \Omega \\
\quad \text{if } p^a_{\mathbf{R}} < \text{len}(s^{a_{\mathbf{R}}}) \\
\quad\quad s^a_{\mathbf{W}} := s^a_{\mathbf{R}} \\
\quad p^a_{\mathbf{I}} := p^a_{\mathbf{I}} + 1 \\
\text{else} \\
\quad \textbf{end}
\end{array}
\tag{28}
$$

10.6.1 Mutation on Copy

In all these experiments, we used two fixed mutation rates. The first of these governs the probability of substitution between the symbol to be copied and a symbol adjacent to it in a *mutation sequence*, as shown in table 2. This probability P_s was set to 0.00001. More rarely still, insertion of an extra random symbol, or deletion of the symbol take place with a much smaller probability $P_i = P_s/(10n)$, where n is the number of different symbol codes. This follows the biology we are trying to emulate. Deletion is simple to emulate. Insertion involves the selection of a random symbol from the entire symbol set.

Mutation on copy depends on comparison of several random numbers r_1 and r_2 with the rates described above.

$$
\begin{array}{l}
\text{if } r_1 < P_i \\
\quad \text{if } r_2 < 0.5 \\
\quad\quad \textbf{INSERT} \\
\quad \text{else} \\
\quad\quad \textbf{DELETE} \\
\text{else} \\
\quad \text{if } r_1 < P_i + P_s \\
\quad\quad \text{if } r_2 < 0.5 \\
\quad\quad\quad \textbf{SUBSTITUTE\_Increment} \\
\quad\quad \text{else} \\
\quad\quad\quad \textbf{SUBSTITUTE\_Decrement}
\end{array}
\tag{29}
$$

The three mutation operations **INSERT**, **DELETE** and **SUBSTITUTE** require a little further explanation. For the **INSERT** operation, a randomly chosen new symbol is inserted at the write pointer - this extends the length of the string by 1 code. The **DELETE** operation is similar - in effect shortening the molecule's string by removing the character that the write pointer indicates. The **SUBSTITUTE** operator makes reference to the substitution matrix ω (see section 6.3) - if **SUBSTITUTE_Increment** is called, the substitution is the next symbol in the mutation sequence described in the matrix relative to the symbol that the read pointer is pointing to. A **SUBSTITUTE_Decrement** call returns the previous symbol in the mutation sequence described in the matrix relative to the symbol that the read pointer is pointing to.

10.7 `%: cleave`

This is used to split the string and generate a new molecule $\mathcal{M}^C$. The string is broken at $p_{\mathbf{F}}$, and the downstream portion is used to create a new molecule. If $p_{\mathbf{F}}$ is at the beginning or end of a molecule, no

action occurs. Any pointers that were pointing at the region downstream of the cleave are moved to $p_{\mathbf{F}}$.

$$\begin{array}{lcll} \mathcal{M}^C : s^1 & := & s_{\mathbf{F}} & \\ s^{\mathbf{F}} & := & s_0^{\mathbf{F}} ... s_{p_{\mathbf{F}}}^{\mathbf{F}} & \\ c & := & a_{\mathbf{F}} & \\ & & & \\ p_{\mathbf{I}}^c & := & p_{\mathbf{F}}^a & \text{if } p_{\mathbf{I}}^c > p_{\mathbf{F}}^a \\ p_{\mathbf{R}}^c & := & p_{\mathbf{F}}^a & \text{if } p_{\mathbf{R}}^c > p_{\mathbf{F}}^a \\ p_{\mathbf{W}}^c & := & p_{\mathbf{F}}^a & \text{if } p_{\mathbf{W}}^c > p_{\mathbf{F}}^a \end{array} \tag{30}$$

There is scope to use modifiers to change which pointer index **`cleave`** splits the molecule at. This is not currently implemented.

10.8 }: `end`

This command terminates execution of the microprogram, and dissolves the bind between the active and passive molecules. The command is also called if $p_{\mathbf{I}} > \text{len}(s)$, which occurs if the instruction pointer is incremented off the end of the string. There are no modifiers to this command, which reverses the binding process described in section 7.4.1. Firstly, the pointers and activators on the active molecule are reset:

$$\begin{array}{lcl} p^1 & := & \perp \\ p^2 & := & \perp \\ s^2 & := & \perp \\ a & := & \perp \end{array} \tag{31}$$

secondly, the status of the two molecules in the pair is set to UNBOUND:

$$\begin{array}{lcl} \mathcal{M}^1 : \delta & := & \text{UNB} \\ & & \\ \mathcal{M}^2 : \delta & := & \text{UNB} \end{array} \tag{32}$$

Note that $\mathcal{M}^2 : s^1$ is not destroyed.

11 Experiments

11.1 Establishing "Invasion when rare" (originally presented in [8])

Here we illustrate the system described above with an experimental evaluation of a replicase metabolism. This experiment was first described in [8]. Inspired by the RNA world model [17], we have conducted studies of metabolic systems composed of molecular microprograms without reference to a genome. In this framework, our molecular species act as their own templates. A replicating molecule is a good candidate for early trials, since only a single species needs to be defined. Our hand-crafted replicase $\mathbf{r}$ is shown in figure 9. Regions 1 and 2 of the microprogram for $\mathbf{r}$ are complementary sequences of match length $l = 7$, match score $s = 5.875$ and $\mathrm{P}\left(\phi\left(\mathbf{r} : s^1, \mathbf{r} : s^1, \omega\right)\right) = 0.293$ We have also hand-crafted a variant of $\mathbf{r}$ that could arise by a single mutation. This variant, $\mathbf{m}$, has perfectly matching complementary sites, so $l = 7$, $s = 7$, $\mathrm{P}\left(\phi\left(\mathbf{m} : s^1, \mathbf{m} : s^1, \omega\right)\right) = 1$, and importantly $\mathrm{P}\left(\phi\left(\mathbf{r} : s^1, \mathbf{m} : s^1, \omega\right)\right) = 1$ also: $\mathbf{r}$ binds to $\mathbf{m}$ more readily than $\mathbf{r}$ binds to $\mathbf{r}$. Note that there are very many species with replicase functionality in our molecular space.

Execution of the microprogram commences at the start of the bind and proceeds stepwise through each symbol to the right. The diagram shows the positions of the executing molecule's pointers (in circles) as the first symbol is about to be copied: $\mathbf{I}$ indicates that the next instruction is **=** (copy). $\mathbf{F}$ is set to the

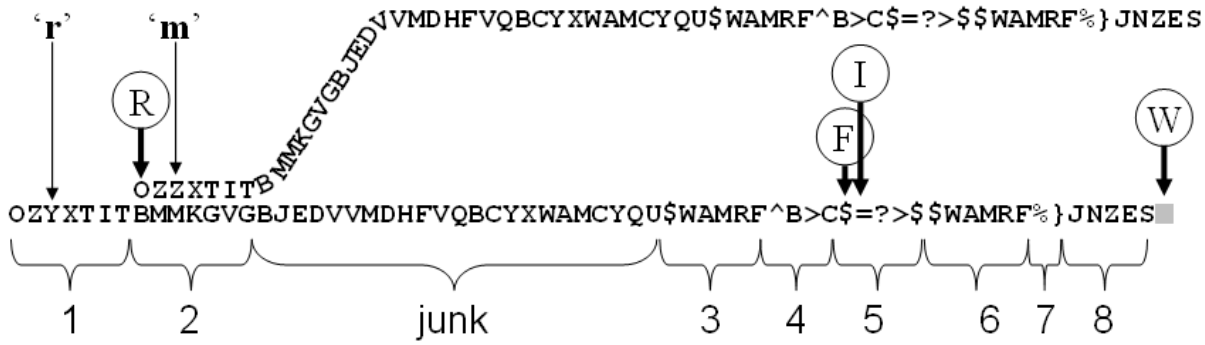


Figure 9: Composition of a replicase enzyme microprogram. The substrate sequence **m**(top) binds to the executing sequence **r** (bottom). There are eight distinct active regions of the molecule, plus one region of "junk" symbols. The sequence specifies: the initial molecular binding (regions 1 and 2); initialisation of the Read (R), Write (W) and Flow (F) pointers (regions 3,4 and 8); iterative copying of the substrate molecule (region 5); Repositioning the flow pointer to the end of the executing molecule (regions 6 and 8); cleaving off the newly created molecule and termination of the microprogram (region 7). Pointers, indicated as letters in circles, show the state of the microprogram as the first symbol of the template molecule is about to be copied to the end of the executing molecule. The two strings differ by a single mutation (indicated by thin arrows), allowing **m** to bind as a substrate to **r** more strongly than **r** binds to a copy of itself.

beginning of region 5, which executes the iterative copy process. $\mathbf{R}$ is positioned at the start of the template molecule's sequence. $\mathbf{W}$ is positioned at the end of the executing molecule's sequence. This is where the new molecule is built.

For experimental trials, we allowed a population of molecule $\mathbf{r}$ to reach equilibrium, then introduced a single molecule of $\mathbf{m}$ at time step 150 000. We ran a metabolism using this specification 100 times. A typical subset of 10 runs is shown in figure 10. The phenomenon of "invasion when rare" is occurring: the competitive advantage of species $\mathbf{m}$ allowed it to replace $\mathbf{r}$ entirely in 88 out of the 100 trials.

11.2 Diversity from a monoculture (originally presented in [18])

We ran 1,000 simulations of a replicase environment under the mutation scheme described above. The goal was to evaluate whether the system would be robust to mutation, and if so, what effects it had on the molecular ecosystem. Each of the 1,000 trials had the potential to run indefinitely and only terminated when there were no molecules remaining in the system. This occurs when the replication mechanism deteriorates in some way so that the replicating molecules cannot copy themselves sufficiently quickly to counter the process of decay. In particular, we sought to identify emergent behaviours in the system that were not part of the original specification and arose by mutation.

11.2.1 The "seed replicase"

Here we describe the molecule used as the seed for the trial. It is one of many possible replicase molecules and is shown in figure 11. There are several features to note:

1. Two binding regions. Two are needed to allow a replicase to bind to a copy of itself because binding is *complementary*: a symbol is a perfect match to a different symbol in the set.

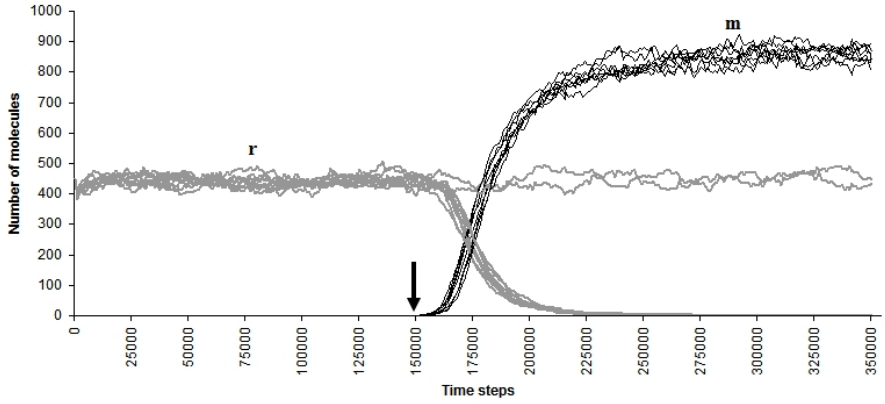


Figure 10: Invasion when rare. Ten typical runs of a metabolism of weakly binding replicase species $\mathbf{r}$ (thick grey lines) which is invaded (indicated by the arrow) by a single molecule of mutant species $\mathbf{m}$ (thin black lines), which has a stronger binding affinity. Eight of these trial runs demonstrate "invasion when rare" by $\mathbf{m}$. In two cases, $\mathbf{m}$ went extinct, and $\mathbf{r}$ remained at equilibrium.

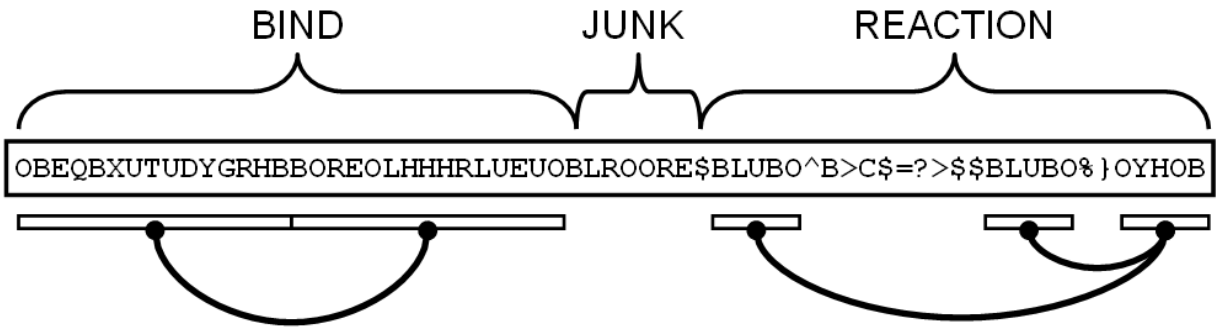


Figure 11: The seed replicase. The top line indicates the regions of the sequence. The sequence itself is shown in the centre box. Complementary alignments are indicated by black connecting lines at the bottom of the figure

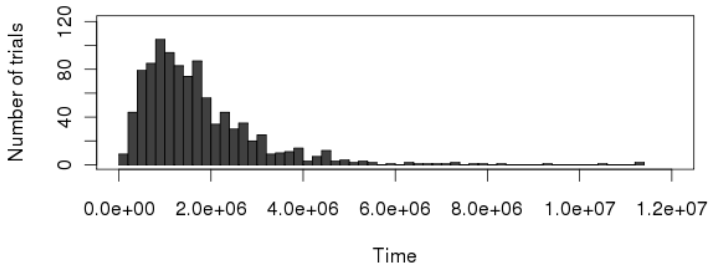


Figure 12: Distribution of extinction times for 1,000 trials

2. A junk region. Mutations here have no effect on the binding or reaction-program, allowing us to explore the effects of neutral mutation *drift*.

3. A functional region. This program specifies that the reaction involves creating a copy of the partner molecule in the reaction.

The seed replicase is 65 instructions long. The reactions takes 240 time steps to construct a new replicase molecule. All of the template codes in the seed replicase are more than one mutation away from a function code. Alignments in the functional region specify program flow. The two binding sites in our seed molecule do not align perfectly, which enables us to evaluate the evolutionary pressure on binding.

11.2.2 Analysis

As part of our evaluation, we developed several ways of representing the simulation data. Each molecule has a sequence of symbols. A particular sequence of symbols denotes a particular molecular species, which has an associated species number. The seed replicase is always species number 1. When a mutation occurs, a molecule with a novel sequence is generated, and this is assigned a new species number. In this way, we can record all new molecular species as they arise. We must also record the dynamics that ensue. Occasionally a new species increases in number and rises to dominance of the system, driving the previous dominant species to extinction. This is known in biology as a *sweep* event. We can capture these events by monitoring when the species number of the most abundant species changes (examples are shown in figure 14). We can record the reactions that exist between all species present in a system at any one time (see figure 16). Finally, we can record the *ancestry* of a molecular species: a new molecule is the product of a reaction between two other molecules, which belong to either one or two species types (see figure 17). These figures are described in more detail later.

With these tools to hand, we are able to demonstrate that our system is capable of producing innovative behaviour even from very simple starting conditions and with no external selection pressure. Essentially, the molecular community acts as a co-evolutionary system, in which the fitness of a particular molecular species is largely determined by the cohort of molecular species with which it shares the container. To demonstrate this, we present results on three levels. The first level gives summary observations and statistics from the 1,000 trials. Secondly, we offer a qualitative analysis of these trials, in which a range of emergent phenomena are qualitatively described. The third analysis gives details of a single trial with emergent phenomena and shows how a series of single-point mutations change the seed replicase system to a mutually-dependent "hypercycle" in which two molecular species cannot self-maintain, but maintain a population by copying each other.

11.2.3 General observations

The mutation rate delivers a mean time of 18,700 time steps for the creation of new molecular species. The majority of these new mutations are not "fixed" in the population and go extinct very quickly. Occasionally a new species arises that has some advantage over the current dominant species.

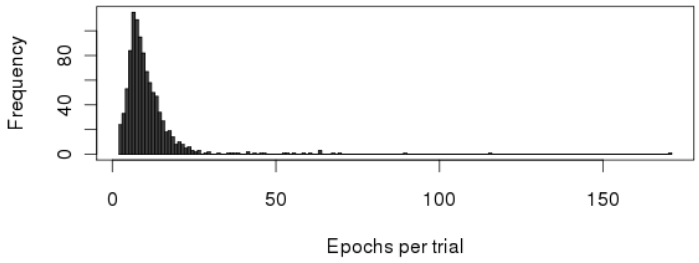


Figure 13: Histogram of number of epochs per trial

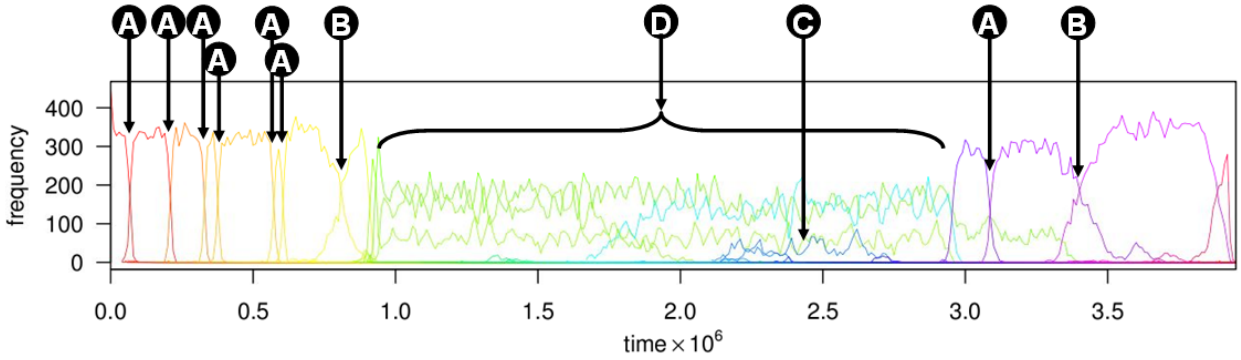


Figure 14: Dominant species in run 112. This trial exhibits (A) characteristic sweeps, (B) slow sweeps, (C) sub-populations, and (D) multispecies hypercycles.

None of the 1,000 trials self-maintains indefinitely. The nature of extinction follows a uniform pattern as described below, but the timing of the extinction varies. Figure 12 shows the distribution of time to extinction for the molecular populations. The modal extinction time is 750000 time steps. In this time an average of 40 new species are produced.

Mutations occasionally produce molecules that rapidly multiply to become the dominant species in the system via the phenomenon of *invasion when rare*. We use the term *epoch* to describe the period over which a particular molecular species is dominant in the system; *sweep* describes a change in epoch. A histogram of the number of epochs per trial is shown in figure 13. The long tail on the histogram is a caused by runs where periods with co-dominant species that should be labelled as a single epoch are recorded by the analysis as a high number of very short epochs due to small fluctuations in abundance of the two species. This definition of the epoch is not particularly useful in situations where two species are co-dominant, but this behaviour was not predicted. Epochs for a single trial can be seen in figure 14.

11.2.4 A classification of emergent phenomena

In this section we give brief descriptions of the key phenomena we have observed in the 1,000 trials. These were identified by visual inspection of the plots of changes in the populations of molecular species, e.g. figures 14 and 15.

Extinction: All trials end when no molecules exist in the system. This occurs when there is a catastrophic decline in replicating molecules. The common cause of this is when a new 'parasitic' molecule arises that is 1) incapable of replicating itself, and 2) copied by the incumbent replicase at a higher rate than the replicase. Note that in order to be copied, a parasite must bind to the replicase sufficiently frequently. This tends to make the system more robust to molecular "junk" and explains why some of the trials continued for so long. A characteristic spike may be observed at the end of each run, which shows this new parasitic molecule as it rapidly increases and then declines when the last replicase molecules decay. Oc-

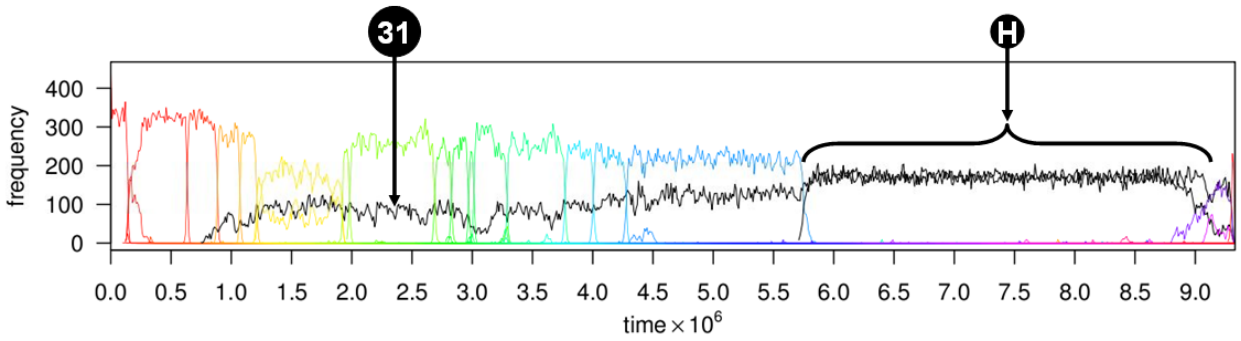


Figure 15: Dominant species in run 277. The short replicase (species 31) emerges at $t = 748,199$ and forms a hypercycle (H) at $t = 5,750,000$.

casionally a parasite begins to overrun the replicase population, but it is unable to bind to a new replicase mutant that is created as the parasitic molecule is increasing. This is rare, occurring in only two of the trials.

Characteristic sweep: The majority of sweeps in our system take a constant form, as shown in figure 14. These are the the main cause of epoch change, and take less than 50,000 time steps for a new mutant to drive the previous dominant species to extinction.

Drift: Drift is observed when a neutral mutation of a dominant individual builds in numbers due to a random walk. Drift is common, occurring in 92 trials. It is plausible that sub-populations and slow sweeps (described below) are both commonly caused by drift. Species exhibiting drift tend to have mutations in the junk region, but can also show mutations in binding regions that do not change the bind affinity.

Sub-populations: These are species which persist in the community in fairly large numbers (more than 50 molecules of approximately 350 in the system). These are very common, occurring in nearly all runs. These sub-populations are nearly always wiped out when a new epoch begins, demonstrating the biological phenomenon of selective sweeps. *Enduring Sub-Populations*, that persist across more than one epoch, occur in 26 trials. This indicates that sub-populations tend to depend on some property of the dominant species in the system, essentially acting as non-lethal parasites. Co-dependence between dominant and sub-populations cannot be determined by examination of population numbers alone. In 2 trials we observed a sweep in a sub-population whilst the dominant population remained stable.

Slow sweeps: A sweep can occasionally take much longer than the 50,000 time steps of a typical sweep. These are called "slow sweeps" and may be due to drift alone. An example can be seen in one of the hypercycle partners in figure 14 at around $t = 2,600,000$. Slow sweeps occurred in 52 trials.

Rapid sweep sequences: Occasionally a mutant causes a "cascade" of new molecules by triggering a sequence of new unseen molecules that quickly dominate the population. The most common mechanism for this is a mutation that gives rise to a series of molecules that bind to a replicase such that less than their entire sequence is copied. This occurs in 31 trials.

Emergent hypercycles: A hypercycle occurs when an enduring sub-population increases in number until it becomes co-dominant with a dominant species. The species forming the enduring sub-population is not self-maintaining, but acts as a copier for the dominant species. The dominant species then repeatedly loses self-self affinity until it loses the ability to self-maintain altogether. The hypercycle occurs when the ability of the dominant population to self-maintain is lost, and the two species become co-dependent. This occurs in 8 trials. Hypercycles end with a sweep, but occasionally one of the partner molecules is still able to maintain a sub-population. A series of sweeps ensues, in which the sub-population declines slightly following each sweep. This occurs in 6 trials.

Spontaneous hypercycles: are the same as the emergent hypercycle, but forms from species that both arise in the immediately preceding epoch. The mechanism is under investigation. This occurs in 15 trials.

Multispecies hypercycles: occur in 14 trials, when there appears to be a mutual dependence among more than two chemical species, as shown in figure 14.

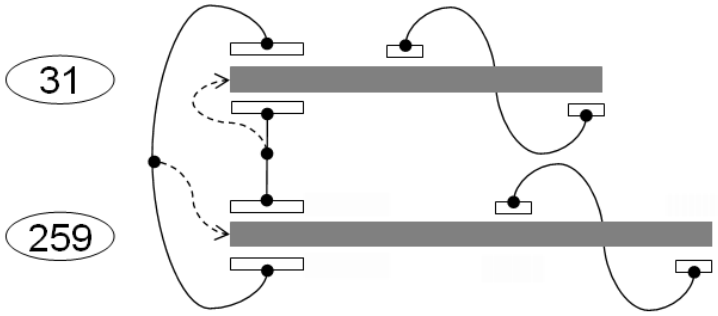


Figure 16: Reactions in the hypercycle. Molecules are represented by grey bars. Binding sites are shown as white boxes, with active binds shown above and passive binds shown below the molecule. Bind alignments are shown as black lines between molecules. Dashed lines show the product of the reaction (where one occurs).

11.2.5 Detailed evaluation of a single trial

We present here details of one of the more interesting sequences of mutation that leads to a hypercycle of co-dependent molecular species. This was observed in trial 277 (figure 15), but hypercycles of one form or another occurred in 30 trials.

We classify this trial as an "emergent hypercycle". At $t = 748,199$ one of the eventual partners (species 31) is first produced via a mutation. This molecule exists as a sub-population for around $5,750,000$ time-steps before forming one partner in a co-dominant pair of molecular species. The partnership runs for approximately 3 million time steps before a parasitic molecule emerges to end the trial.

The molecular species in a hypercycle: The two molecular species (31 and 259) in the hypercycle are shown in figure 16. The bindings that occur between them are shown as black lines. The assignment of roles in the reaction (i.e. whether the molecule is passive (acts as the template) or active (acts as the program) occurs with equal probability for both molecules, meaning that for 50% of the time species 31 is produced and for the other 50% of the time species 259 is produced. Also note that species 31 is shorter than species 259 - it has lost one of the binding regions required for the reaction-program to initialise such that a copy of the replicase is created. This means it tends to be copied more quickly. Neither molecule is able to self-copy.

This phenomenon was neither foreseen in the original design nor expected to form without further design effort. It is particularly surprising that both partners in our hypercycle have no ability to self-copy. How could this have happened, and what is the evolutionary advantage of it?

Origin of the short partner: We need to explain how species 31, that is missing a key functional component, can rise to co-dominance in our system. We can trace the ancestry of the molecular species, and examine the reaction networks at key stages in any trial (figure 17). A white box indicates that a new species is synthesised *de novo* in the reaction, whereas a grey box indicates that the new species arises by modification of one of the reactants. Replicase molecules should act as catalysts, remaining unchanged when they emerge from a reaction. We can conclude that there is something in the reaction with molecules of species 29 that has produced species 30, which then reacts with species 9 to form species 31. The single point mutation of species 9 to create species 29 is shown below by a vertical line:

```
009 OBEQBX...LHHHRLUEUOBLROORE$BLUBOˆB>C$=?>$$BLUBO%}OYHOB
                                 |
029 OBEQBX...LHHHRLUEUOBLROORE$BLUBPˆB>C$=?>$$BLUBO%}OYHOB
```

The subsequence `$BLUBO` has mutated to `$BLUBP`. The `$` symbol is a code for "seek", and (in this situation) positions the molecule's flow pointer at the end of the best complementary alignment for the sequence `BLUBO`, which is the sequence `OYHOB`. With the mutation in species 29, the alignment spans only

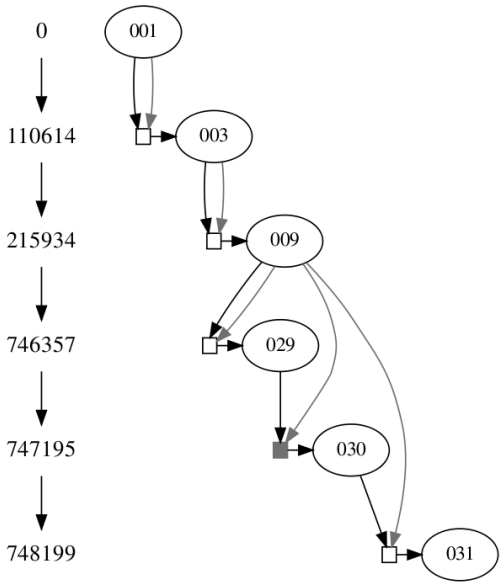


Figure 17: Ancestry of species 31. Numbers on the left indicate the time of reaction. Black arrows indicate the active partner. Grey arrows indicate the passive partner

```
030 OBEQBXUUUDYGRHBBOSEOLHHHRLUEUOBLROORE$BLUBPˆB>C$=?>$$BLUBO%}OYHOOBEQBXUUUDYGRHBBOSEOLHHHRLUEUOBLROORE$BLUBOˆB>C$=?>$$BLUBO%}OYHOB
Bind site:                                                |--------------|
009                                                 OBEQBXUUUDYGRHBBOSEOLHHHRLUEUOBLROORE$BLUBOˆB>C$=?>$$BLUBO%}OYHOB
Product:                                                  |-----------------------------------------------|
031                                                       BBOSEOLHHHRLUEUOBLROORE$BLUBOˆB>C$=?>$$BLUBO%}OYHOB
```

Figure 18: Origin of species 31

the first four letters of `$BLUBO`, so the copy of the molecule is constructed one symbol in from the end of the molecule. When the construction is complete, the newly-created string must be cleaved from the active molecule's sequence. The pointers are arranged to achieve this via a second "seek" command with the same target (`OYHOB`). However, since the target has been overwritten in the original molecule, the seek command positions the pointer at the *end* of the newly copied molecule instead. The "cleave" command is applied to the far end of the string and is thus ineffective. The reaction-program terminates, and the new molecule (species 31) is created from most of a molecule of species 29 with a copy of species 9 pasted over the penultimate symbol.

In this manner, the reaction between species 29 and 9 creates species 30, which is nearly twice as long as the seed replicase, as shown in figure 18. Note there is only ever a single molecule of species 29, which is immediately transformed into species 30 when it reacts with a molecule from species 9. When species 9 binds to species 31, the bind site is shifted to a new position, as shown in figure 18. This changes the action of the replicase program such that the first 14 characters of the string are not copied. In this way, the single instance of species 30 can create many molecules of species 31 until it decays. Species 31 is then copied by dominant species in the system in 50% of reactions with it. Note that this cascade of reactions all occurs as a result of the single-point mutation on species 9.

Evolutionary pressure towards a hypercycle: Having established how a shorter molecule can arise via single-point mutations, we need to investigate how the molecule persists in the system, and what evolutionary pressure there is towards the formation of a hypercycle. It is important to note that in our replicase system a molecule that ensures it will always act as the template in a reaction is likely to sweep the population, as it will increase in numbers whenever it binds to another molecule. This is often achieved by *reducing* the bind probability for self-self reactions: as long as a bind is sufficiently likely, all the energy available in the system can be consumed. Binds stronger than this critical value have no advantage, whereas increasing

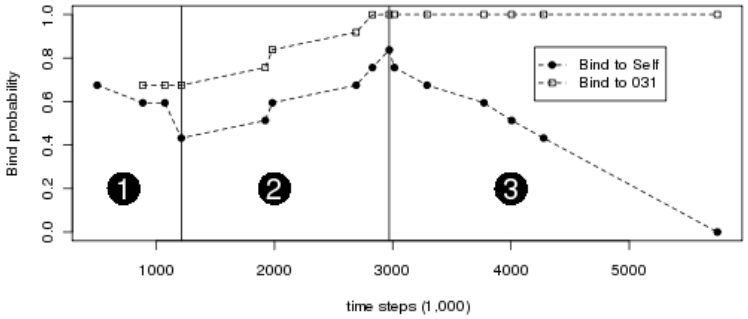


Figure 19: Change in binding rates as a precursor to hypercycle emergence

any bias towards becoming the template in a reaction is clearly advantageous. For single-replicase systems, this is straightforward to understand, but with the introduction of species 31, the dynamics get more interesting.

Once present in the system, species 31 becomes a resource for other molecules. In all of the reactions with species 31, the chances of acting as a template are 50-50 (since the position of the alignment is the same on each string). This means that new species that bind to 31 can use it as a resource for increasing their number, even though half the time they will be exploited by species 31 to maintain its own population. Through a series of sweeps, each new dominant species binds increasingly strongly to species 31, thus flushing the previous incumbent from the system. Any new species that binds *less* strongly to species 31 than the previous dominant species is unsuccessful: it loses in the competition to exploit a valuable resource. Once bind affinity to species 31 is maximised, the old strategy of weakening self-self binds to guarantee template status in a reaction takes over again.

These processes are illustrated in figure 19, which plots binding rates for new dominant species in trial 277. The plots show the changes in bind probabilities with each successive sweep of the population as illustrated in figure 15. The line labelled "Bind to self" shows the probability of self-self binding for each new dominant species. The line labelled "Bind to 31" shows the bind probability between the new dominant species and species 31. There are three phases. The first phase shows a decrease in self-binding probability between successive dominant species. We then see a second phase in which new species have an increasing affinity for binding to molecule 31. Once this is maximised, the third phase begins, in which successive dominant species sacrifice their self-bind probability to ensure they act as templates when reacting with the previous dominant species. In this way, dependence upon species 31 increases, until self-replication disappears altogether, and a hypercycle emerges.

The single-point mutations between dominant species are shown in figure 20. It shows that all mutations that confer an advantage occur in the binding regions of the molecule. Phases 1 and 3 of the run show changes in the second bind region, whereas phase 2 shows mutations in the first bind region. This corresponds with the change in phase noted for figure 19. The functional region of the molecule, which occupies the last half of the string, is preserved throughout. This is far from a random walk: the critical function of the replicase is preserved throughout, whilst a continual turnover of the binding site sequences illustrates the evolutionary pressure on the molecular species to act as a template for the molecule that the replicase builds.

11.2.6 Conclusions

We have presented an evaluation of the effect of mutation on an open-ended chemical system. The richness of behaviour we have shown is striking; indeed it was unexpectedly rich given that the only form of mutation is single-point. The need for such richness in complex systems was one of our main considerations during the design of this system. In addition, our chemistry reveals something of the dynamics of replicase systems

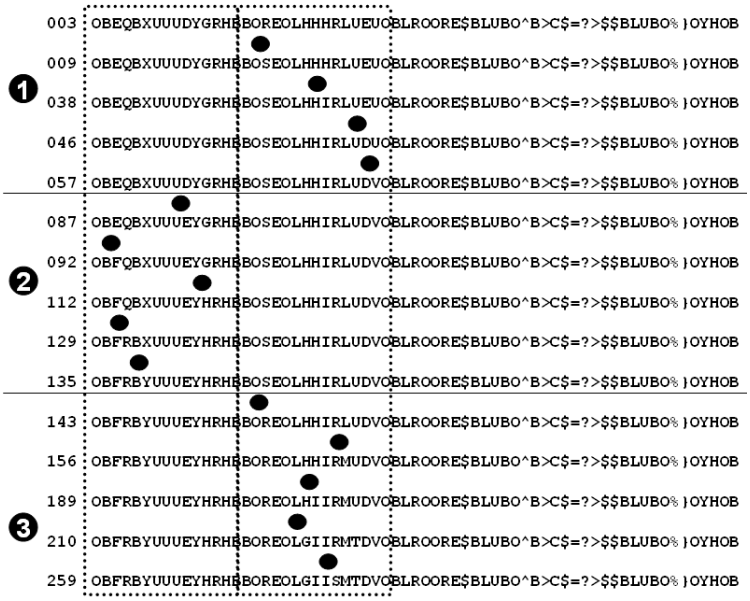


Figure 20: Mutations for the dominant species in run 277. Bind sites are indicated with dashed lines.

that is very difficult to observe in biology. The decrease in binding affinity was not predicted, and the mechanism by which the hypercycle emerged was the result of a macromutation that was not "designed in" to the system.

Our replicase molecules are "imperfect replicators": they have a small chance of making an error when copying anything that binds to a certain region on the molecule. The imperfections in the copy process are not currently encoded on the genome; they are preset in the microcode of the copy instruction and thus unavailable for manipulation on the genome. In future work, we could represent the copy instruction at a finer level of granularity and use template codes to specify the accuracy of each sub operation, possibly including some cost for an increased accuracy of copy. We observed macro-mutations arising as a result of single-point changes that delivered emergent phenomena due to the wide heritable range of the system.

Finally, we must emphasise that these trials form a control experiment in which the effects of single-point mutation were evaluated. Future work will examine the effects of running a "population" of these trials, such that when a population of molecules collapses in an individual container, it can be replenished by a neighbour. This gives us a full model of early life, in which replicating templates and machinery self-maintain within membrane-bounded containers that can be replenished by neighbours.

12 Conclusion

This document describes the first version of an artificial chemistry which allows binding and reaction function to be described by a sub-molecular sequence. Most of our efforts have been spent on defining a "baseline chemistry" and delimit the ingredients of the cell model and the generic reaction events. This has allowed us to build a modular cell model in which a chemistry appropriate for evolution can be developed.

We have shown that a replicator molecule can be built within this chemical model by building a replicase metabolsim. Although this is a useful first step, we now need to enlarge the model to accommodate more ambitious simulations.

References

[1] Hickinbotham, S., Clark, E., Stepney, S., Clarke, T., Young, P.: Gene regulation in a particle metabolome. In: CEC 2009, Trondheim, Norway, May 2009, IEEE Press (2009) 3024–3031

[2] Fischer, V., Hickinbotham, S.: A metabolic subsumption architecture for cooperative control of the e-puck. In: NICSO. (2010) accepted

[3] Clark, E., Hickinbotham, S., Stepney, S., Clarke, T., Young, P.: Encoding evolvable molecules. In: International Workshop on Information Processing in Cells and Tissues (IPCAT), Franscini Ascona, Switzerland, Librix (2009)

[4] Bentley, P.J.: Fractal proteins. Genetic Programming and Evolvable Machines (2004) 71–101

[5] Spiegelman, S., Kacian, D.L., Mills, D.R., Kramer, F.R.: A replicating rna molecule suitable for a detailed analysis of extracellular evolution and replication. Proceedings of the National Academy of Sciences **69** (1972) 3038–3042

[6] Johnson, T.J., Wilke, C.O.: Evolution of resource competition between mutually dependent digital organisms. Artif. Life **10** (2004) 145–156

[7] Ray, T., Xu, C.: Measures of evolvability in tierra. Artificial Life and Robotics **5** (2001) 211–214

[8] Hickinbotham, S., Clark, E., Nellis, A., Pay, M., Stepney, S., Clarke, T., Young, P.: Molecular microprograms. In: ECAL 2009 (LNCS 5777), Springer (in press)

[9] Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. J Mol Biol **147** (1981) 195–197

[10] Dill, K.A., Ozkan, S.B., Shell, M.S., Weikl, T.R.: The protein folding problem. Annual Review of Biophysics **37** (2008) 289–316

[11] Pennock, R.T.: Models, simulations, instantiations, and evidence: the case of digital evolution. J. Exp. Theor. Artif. Intell. **19** (2007) 29–42

[12] Ierymenko, A.: Nanopond: A very tiny artificial life virtual machine. `http://adam.ierymenko.name/nanopond.shtml` (2010)

[13] Bobrik, M., Kvasnicka, V., Pospichal, J.: Artificial chemistry and molecular Darwinian evolution of DNA/RNA-like systems I – typogenetics and chemostat. In Kelemen, A., Abraham, A., Liang, Y., eds.: Computational Intelligence in Medical Informatics. Volume 85 of Studies in Computational Intelligence. Springer (2008) 295–336

[14] You, L.: Toward computational systems biology. Cell Biochem Biophys **40** (2004) 167–184

[15] Shpaer, E.G., Robinson, M., Yee, D., Candlin, J.D., Mines, R., Hunkapiller, T.: Sensitivity and selectivity in protein similarity searches: A comparison of smith-waterman in hardware to blast and fasta. Genomics **38** (1996) 179 – 191

[16] Wikipedia: Smith-waterman algorithm, Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Smith-Waterman_algorithm` (2009) [Online; accessed 14-April-2009].

[17] Lincoln, T.A., Joyce, G.F.: Self-sustained replication of an RNA enzyme. Science (2009) 1167856+

[18] Hickinbotham, S., Clark, E., Stepney, S., Clarke, T., Nellis, A., Pay, M., Young, P.: Diversity from a monoculture: Effects of mutation-on-copy in a string-based artificial chemistry. In: ALife XII, Odense, Denmark, August 2010, MIT Press (2010) 24–31

A Version control summary

Version 0.2 – issued 1 May 2010

Changes from v0.1:

- The decay protocol for molecules was changed so that they can decay when bound (section 4)
- The way the bind probability is calculated from the alignment score was changed (section 6.5)
- The concept of mutation was introduced as an addition to the **`copy`** operator (section 10.6.1)

This is the version used in the paper [18]

Version 0.1 – issued 1 January 2010

This is the baseline version, used in paper [8].

B Worked example: processing string alignments

Section 6.4 described how our molecular analogues bind via a process of complementary Smith-Waterman alignment. To illustrate the process, let us consider the alignment of two strings:

s' : **NNNOUV>$PQVONNN**
$C(s')$: **AAABHG>$CDGBAAA**
s : **ABABABHG$CDBAAC**

table 6 shows the matrices H and T for this alignment. The maximum value is 7.72, which is the *score* σ of the alignment. We have underlined this value, and underlined the trace back through the matrix by which the alignment is found. The trace through T is shown with large arrows.

			A	A	A	B	H	G	>	$	C	D	G	B	A	A	A
s'	$C(s')$																
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
N	A	0	1.00	1.00	1.00	0	0	0	0	0	0	0	0	0	1.00	1.00	1.00
O	B	0	0	0.88	0.88	2.00	0.67	0	0	0	0	0	0	1.00	0	0.88	0.88
N	A	0	1.00	1.00	1.88	0.76	0.88	0	0	0	0	0	0	0	2.00	1.00	1.88
O	B	0	0	0.88	0.88	2.88	1.55	0.22	0	0	0	0	0	1.00	0.67	1.88	0.88
N	A	0	1.00	1.00	1.88	1.55	1.76	0.55	0	0	0	0	0	0	2.00	1.67	2.88
O	B	0	0	0.88	0.88	2.88	1.55	0.88	0	0	0	0	0	1.00	0.67	1.88	1.55
U	H	0	0	0	0	1.55	3.88	2.55	1.22	0	0	0	0	0	0	0.55	0.76
T	G	0	0	0	0	0.22	2.55	4.88	3.55	2.22	0.89	0	1.00	0	0	0	0
$	$	0	0	0	0	0	1.22	3.55	4.38	4.05	2.72	1.39	0.03	0.75	0	0	0
P	C	0	0	0	0	0	0	2.22	3.05	4.26	5.05	3.72	2.39	1.06	0.50	0	0
Q	D	0	0	0	0	0	0	0.89	1.72	2.93	4.14	6.05	4.72	3.39	2.06	0.73	0
O	B	0	0	0	0	0	0	0	0.39	1.60	2.81	4.72	5.17	5.72	4.39	3.06	1.73
N	A	0	1.00	1.00	1.00	0	0	0	0	0.27	1.48	3.39	3.84	5.05	6.72	5.39	4.06
N	A	0	1.00	2.00	2.00	0.88	0	0	0	0	0.15	2.06	2.51	3.72	6.05	7.72	6.39
P	C	0	0	0.75	1.75	1.88	0	0	0	0	1.00	0.73	1.31	2.39	4.72	6.39	7.47
N	A		·	·	·										·	·	·
O	B			↖	↖	↖	←							·		↖	↖
N	A		·	·	↖	↖	↖								↖	·	↖
O	B			↖	↖	↖	←	↖						·	↑	↖	↖
N	A		·	·	↖	↑	↖	↖							↖	↖	↖
O	B			↖	↖	↖	↖	↖						·	↑	↖	↖
U	H					↑	↖	↖	←							↖	↖
T	G					↑	↖	↖	←	↖	←		·				
$	$						↖	↖	↖	↖	←	←	←	↖			
P	C							↖	↑	↖	↖	↖	↖	←	←		
Q	D							↑	↑	↖	↖	↖	←	↖	↖	←	
O	B								↑	↖	↑	↖	↖	↖	↖	↖	↖
N	A		·	·	·					↑	↖	↑	↑	↖	↖	↖	↖
N	A		·	↖	↖	↖					↑	↑	↑	↖	↖	↖	↖
P	C			↖	↖	↖					·	↑	↖	↖	↖	↖	↖

Table 6: A subsequence alignment H (shown top) generated from the scoring matrix in table 3. The alignment trace matrix T (bottom) contains all possible alignments. The final resulting alignment is shown with large arrows.

The trace through H and T produces the following alignment:

s' : **--NNNOUV>$PQVONNn**
$C(s')$: **--AAABHG>$CDGBAAa**
s : **abABABHG-$CD-BAAc**

where characters outside the alignment are shown in lower case, and indels are indicated by a dash. Note also that the length of the alignment of each string is different, due to the indels.

B.1 Note regarding implementation of Smith-Waterman alignments in version 0.1

It has come to our attention that the original implementation of the Smith-Waterman algorithm was not accurate because it did not use the trace-back matrix T to find the start of the alignment from the end position. Instead, the original implementation sought a route from the end point which followed the highest value in the cells to the left, above and above-left of the cell currently under consideration. This method has the advantage that the trace matrix T does not need to be calculated, but it produces different alignments which are most noticeable with short repeats of code sequences.

The effects of this difference can be best understood by example. Taking the matrix shown in figure 6, we can trace a new path using the "highest value" approach. This pathway is shown in red in figure 7.

			A	A	A	B	H	G	>	s $	C	D	G	B	A	A	A
s'	C(s')																
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
N	A	0	1.00	1.00	1.00	0	0	0	0	0	0	0	0	0	1.00	1.00	1.00
O	B	0	0	0.88	0.88	2.00	0.67	0	0	0	0	0	0	1.00	0	0.88	0.88
N	A	0	1.00	1.00	1.88	0.76	0.88	0	0	0	0	0	0	0	2.00	1.00	1.88
O	B	0	0	0.88	0.88	2.88	1.55	0.22	0	0	0	0	0	1.00	0.67	1.88	0.88
N	A	0	1.00	1.00	1.88	1.55	1.76	0.55	0	0	0	0	0	0	2.00	1.67	2.88
O	B	0	0	0.88	0.88	2.88	1.55	0.88	0	0	0	0	0	1.00	0.67	1.88	1.55
U	H	0	0	0	0	1.55	3.88	2.55	1.22	0	0	0	0	0	0	0.55	0.76
T	G	0	0	0	0	0.22	2.55	4.88	3.55	2.22	0.89	0	1.00	0	0	0	0
$	$	0	0	0	0	0	1.22	3.55	4.38	4.05	2.72	1.39	0.03	0.75	0	0	0
P	C	0	0	0	0	0	0	2.22	3.05	4.26	5.05	3.72	2.39	1.06	0.50	0	0
Q	D	0	0	0	0	0	0	0.89	1.72	2.93	4.14	6.05	4.72	3.39	2.06	0.73	0
O	B	0	0	0	0	0	0	0	0.39	1.60	2.81	4.72	5.17	5.72	4.39	3.06	1.73
N	A	0	1.00	1.00	1.00	0	0	0	0	0.27	1.48	3.39	3.84	5.05	6.72	5.39	4.06
N	A	0	1.00	2.00	2.00	0.88	0	0	0	0	0.15	2.06	2.51	3.72	6.05	7.72	6.39
P	C	0	0	0.75	1.75	1.88	0	0	0	0	1.00	0.73	1.31	2.39	4.72	6.39	7.47
N	A		·	·	·										·	·	·
O	B			↖	↖	↖	←							·		↖	↖
N	A		·	·	↖	↖	↖								↖	·	↖
O	B			↖	↖	↖	←	↖						·	↑	↖	↖
N	A		·	·	↖	↑	↖	↖							↖	↖	↖
O	B			↖	↖	↖	↖	↖						·	↑	↖	↖
U	H					↑	↖	↖	←							↖	↖
T	G					↑	↖	↖	←	↖	←	·					
$	$						↖	↖	↖	↖	←	←	←	↖			
P	C							↖	↑	↖	↖	↖	↖	←	←		
Q	D							↑	↑	↖	↖	↖	←	↖	↖	←	
O	B								↑	↖	↑	↖	↖	↖	↖	↖	↖
N	A		·	·	·					↑	↖	↑	↑	↖	↖	↖	↖
N	A		·	↖	↖	↖					↑	↑	↑	↖	↖	↖	↖
P	C			↖	↖	↖					·	↑	↖	↖	↖	↖	↖

Table 7: "Highest score" alignment. A subsequence alignment H (shown top) generated from the scoring matrix in table 3. The alignment trace matrix T (bottom) contains all possible alignments. The alignment is shown in large red arrows. For comparison, the alignment trace using Smith-Waterman is shown in large black arrows.

The trace through H and T shown in red produces the following alignment:

$$
\begin{aligned}
s' &: \quad \texttt{NNNOUV>\$PQVONNn} \\
C(s') &: \quad \texttt{AAABHG>\$CDGBAAa} \\
s &: \quad \texttt{ababA--BHG\$C-D-BAAc}
\end{aligned}
$$

This can be intuitively seen to be suboptimal when compared with the alignment shown on the previous page.

A full Smith-Waterman alignment method is used in the current version of the Stringmol software. We have demonstrated empirically that the rich phenomena seen using version 0.1 is still producable using this new version, but we have had to use different sequences to produce it. This is because it appears to be

critically important to use a sequence in which part of the alignment changes with the addition or removal of codes at the start of the alignment.

Using Smith-Waterman, a seed replicase showing similar phenomena to that described in [18] has the following sequence:

```
OOGEOLHHHRLUEUOBBBRBXUUUDYGRHBLROORE$BLUBOˆB>C$=?>$$BLUBO%}OYHOB
```

Where a mutation on the **BLUBO – OYHOB** alignment occurs the program attempts to align **BLUBP – OYHOB**. A double length molecule is created, similar to the process described in section 11.2.5. The key point is that the sequence **OOGEOLHHHRLUEUO** aligns optimally at the point where the molecule repeats, in turn causing the short molecule to be created. See section 11.2.5 for further details.

C Experiments with molecular decay

This section records experiments carried out to test the protocol for molecular decay. There are two aspects to the protocol. The first is the rate at which decay occurs. The second is the state in which the molecule must be to be available to decay. The original decay protocol was reported in [8], where the molecules were not subject to mutation. It was found that once mutation was introduced into the system, the decay protocol caused unsatisfactory mutational trajectories to arise.

In [8], the decay rate was set to be inversely proportional to the length of the molecule (hereinafter referred to as "DFL"). In addition, it was specified that decay could only occur whilst a molecule was unbound ("UNB"). Here we evaluate alternatives to these protocols: a constant decay rate ("DC") and no requirement that decay occur only whilst the molecule is unbound - decay can occur no matter what state the molecule is in ("ALL"). If decay occurs whilst a molecule is bound, then the other molecule participating in the bind is destroyed.

Evolutionary pressure of Decay protocols

Once mutation was introduced, it was found that the decay protocol placed unforseen evolutionary pressures on the system. DFL confers a huge evolutionary advantage on longer molecules, and so we should predict ever increasing numbers of long molecules in the system, since longer molecules persist in the system and are thus available for copying over a longer period of time. Similarly, specifying that decay can only occur whilst a molecule is unbound (UNB as opposed to ALL) selects for molecules which remain in a reaction permanently (via the emergence of an infinite while loop).

Evaluation

Figure 21 show the effects of combining these strategies. DFL + UNB forces the evolutionary trajectory towards immortal molecules with infinite while loops. Ever-increasing numbers of a particular molecule appear in the system even though energy is limited. The processing overheads of these scenarios are unbounded, and the chemistry is not particularly rich. A similar situation occurs with DFL + ALL, but rather than having one particular species dominate, a large number of molecular species with very little functional variation between them arises. Here the number of molecules in the system gets very large as the length of the molecules increases. Although this is number is bounded by the decay rate of the longest possible molecule, it is difficult to monitor any evolutionary change. Trials in which the decay rate is constant (DC) tend to have more stable populations. However the UNB specification tends to influence the evolution of molecular species with non-terminating reactions. The result is that the population of molecules reduces to a set of pairs of molecules participating in non-terminating reactions. No further evolution of the system can be observed. Only by allowing decay of molecules in all states can this situation be avoided.

However, if we combine the "DC" and "ALL" decay protocols, we observe a regular turnover of dominant molecular species, which is a requirement for our studies in evolution of chemical systems.

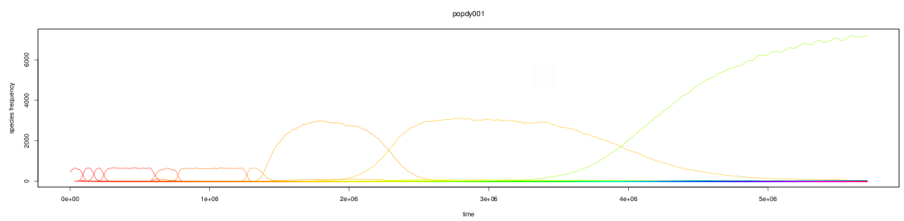
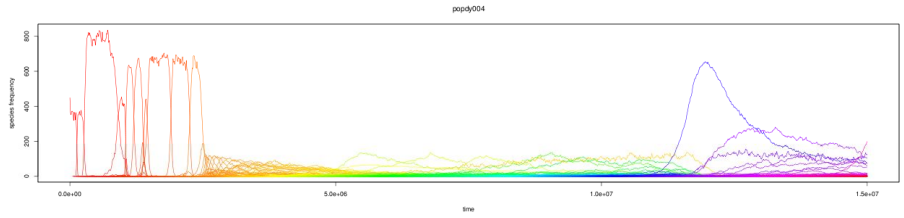
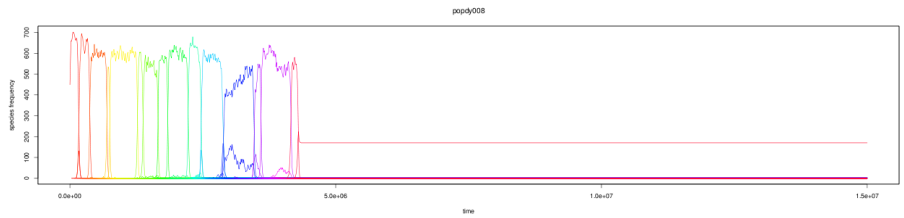
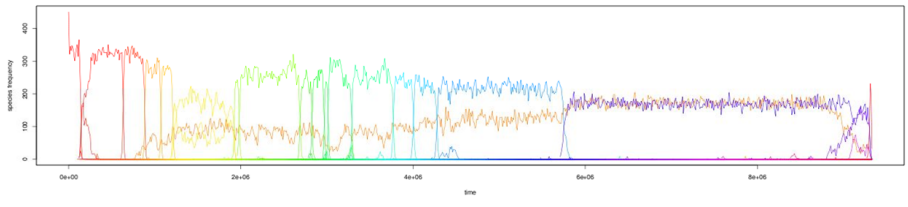
Rate function	State	Publication	Example trial
DFL	UNB	[8]	
	ALL	_	
DC	UNB	_	
	ALL	[18]	

Figure 21: Sample trials for different decay regimes.

D Evaluation of binding strategies

Binding sites are regions of molecules that have sequences which (through a process described in detail in section 6) match some other sequence in another molecule in the system. The match strength determines the chance of a pair of molecules reacting together, and is a function of the length of the alignment and the level of matching between the symbols in the aligned sequence. This is a powerful biological analogy, but a difficult design challenge.

The replicase system presented in [8], based on version 0.1 of the Stringmol Environment has been shown to be self-maintaining in a simple environment where there is no mutation. Having introduced mutation on copy to the system (as described in section 10.6.1 of the current version), it was found that there were some deficiencies in the binding scheme: Single-character alignments gave bind probabilities of 1.0; for alignments of length greater than 1, only perfect alignments could score 1.0; single-character mismatches drastically reduced the bind probability of long alignments. It was clear that a new mapping between the length and score of alignments to a bind probability was desirable. Here, we compare the effect of mutation on binding in environments with:

1. Molecules with short and long binding sites.
2. Molecules with different symbols in the binding sequence.
3. Weakly-binding molecules.
4. Hypercycles of mutually copying molecular species

These systems will be tested with the two different equations for calculating the bind probability as presented in versions 0.1 and 0.2 of this technical report.

D.1 The molecular zoo

To evaluate binding strategies, we devised six replicase configurations, differing only in their binding sites. Table 8 shows these molecules and their binding sites. There are three classes of replicase in this set, indicated by three prefixes. Molecules with the prefix 'R1' have binding sites consisting of symbols from the set of all available template codes. Molecules with the prefix 'R2' have binding sites consisting of symbols that are more than one mutation away from mutation into a functional symbol. Molecules with the prefix 'Hyp' form part of a hypercycle system in which two molecular species maintain their population by copying each other. Postfixes 'short' and 'long' are intuitive labels indicating whether the binding site is short (7 symbols) or long (15 symbols). The postfix 'weak' indicates a molecule with a long binding site which has a low chance of binding.

These molecular specifications allow us to test three hypotheses: **1** that the characteristics of the bind sites (length, strength, and distance from lethal mutations) have an effect on the frequency of an adaptive mutation; **2** that species with weak binds are out-competed by species with strong binds; **3** that hypercycles have similar sensitivities to mutation compared with single-copy systems.

We use two measures of fitness in these trials: time to extinction and number of epochs. These metrics are straightforward to calculate, but rather crude. Future work on the Stringmol system will help to create more sensitive evaluation strategies.

Trial	Code Set	Site Length	Bind probability	Name	Sequence
1	1	7	1.0	R1Short	OZYXTITBMMKGVGBJEDVVMDHFVQBCYXWAMCYQU$WNZRF^B>C$=?>$$WNZRF%}JAMES 1 2
2	1	15	1.0	R1long	OZYXTITBMMKGVGBBMMKGVGOZZXTITOWAMCYQU$WNZRF^B>C$=?>$$WNZRF%}JAMES 1 2
3	2	7	1.0	R2Short	OBDRBYUBOREOLHBJEDVVMDHFVQBCYXWAMCYQU$WNZRF^B>C$=?>$$WNZRF%}JAMES 1 2
4	2	15	1.0	R2long	OBDRBYUUUEYHRHBBOREOLHHHRLUEUOWAMCYQU$WNZRF^B>C$=?>$$WNZRF%}JAMES 1 2
5	2	15	weak	R2weak	OBEQBXUTUDYGRHBBOREOLHHHRLUEUOWAMCYQU$WNZRF^B>C$=?>$$WNZRF%}JAMES 1 2
6	1	7	1.0	Hyp1	OZZXTITBOREOLHBJEDVVMDHFVQBCYXWAMCYQU$POXGL^B>C$=?>$$POXGL%}CBKTY 1a 2b
6	1	7	1.0	Hyp2	OBERBYUBMMKGVGBJEDVVMDHFVQBCYXWAMCYQU$WNZRF^B>C$=?>$$WNZRF%}JAMES 1b 2a

Table 8: Molecular species used in the binding trials

D.2 Bind strength equations

We have two approaches to obtaining a bind probability from the Smith-Waterman score. The first P_1, from **??** was:

$$\mathrm{P}_1(\phi(s,s',\omega)) = \begin{cases} 0 & \text{if } l <= 3 \\ (\sigma/\lambda)^{\lambda} & \text{otherwise} \end{cases} \tag{33}$$

Where $P(B)$ is the probability of binding, s is the Smith-Waterman score for the alignment and l is the alignment length. This was our initial formulation and was designed to reward highly-scoring bind. The problem with this is that for longer binds, the score tends to be weaker since the fraction is raised to the power l. Very short binds can bind just as strongly as very long ones. A small number of mutations away from a perfect match can have a highly detrimental effect on long binding sites, rather than having the effect of "tuning" the bind probability to a rate appropriate to the metabolism of the system. There is also little redundancy at the top end of the bind probability profile - we would like there to be many ways of achieving a bind probability of 1. We therefore tried a the following binding probability equation:

$$\mathrm{P}_2(\phi(s,s',\omega)) = \begin{cases} 0 & \text{if } \lambda <= 3 \\ \min(\sigma, \lambda - m)/(\lambda - m) & \text{otherwise} \end{cases} \tag{34}$$

(equation 13 in the main body of this report) where m is the penalty for a single-point mismatch in the Smith-Waterman alignment matrix.

D.3 Experiments

We ran six trials for both binding schemes. The trials used a mutation scheme described in section 10.6.1. Each trial consisted of twenty runs of a metabolism. The runs lasted 1,500,000 time steps or until there were no molecules in the system (extinction). The products of each reaction were checked to see if a new molecular species (as specified by the sequence of symbols representing the molecule) had been created. Five of the six trial types were uni-molecular systems consisting of a single replicase species. The exception to this was the hypercycle trial, which consisted of two molecular species whose population was maintained by reciprocal copying. The seed replicase molecules used in these experiments are shown in table 8.

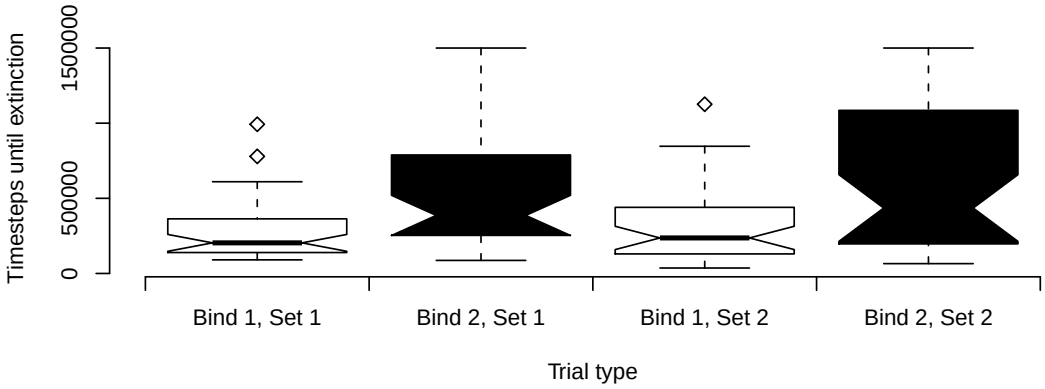


Figure 22: Effect of bind encoding on extinction time

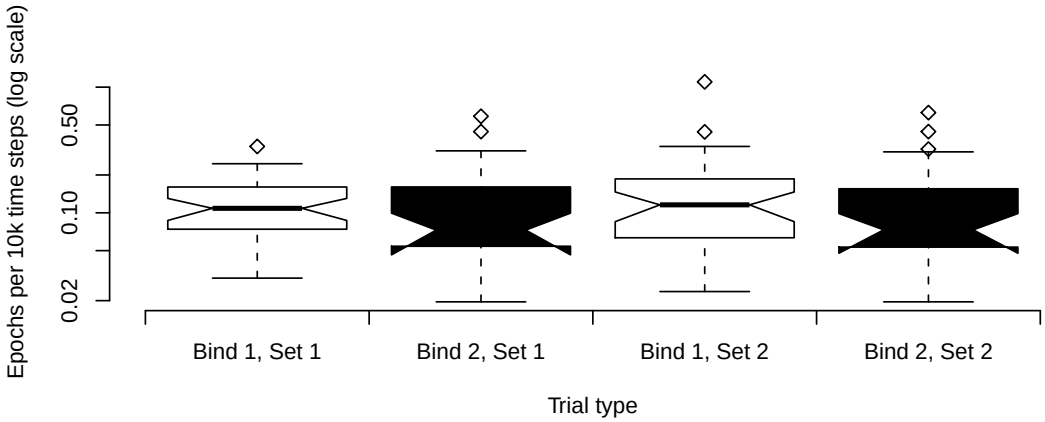


Figure 23: Effect of bind encoding on the number of epochs per 10,000 time steps

D.4 Results

D.4.1 Bind encodings

Figures 22 and 23 show the effects of the two binding schemes and the symbols used to encode binding sites on the time to extinction and the number of epochs per 10^4 time steps respectively. Here it can be seen that trials using the the new bind scheme (shown in black in figures 22 to 29) tend to run for slightly longer, but show no greater turnover of dominant species. A possible explanation for this is that there is scope for neutral mutation between strong bind configurations, since a single mis-match still scores 1.0 in the new bind scheme.

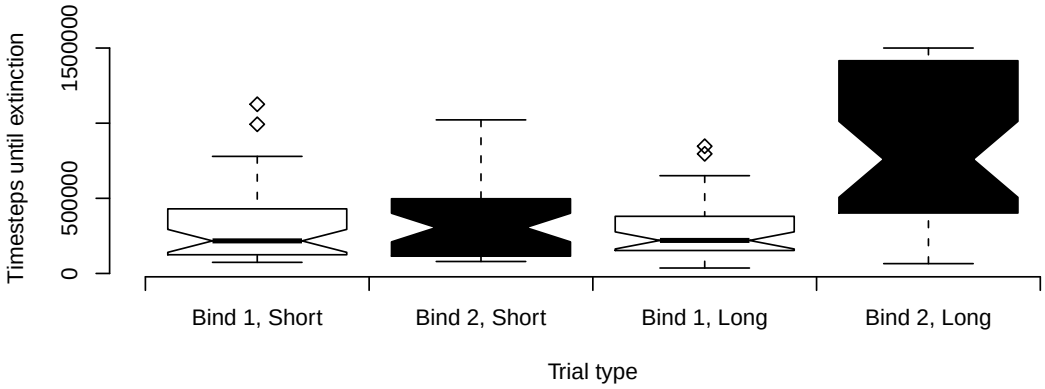


Figure 24: Effect of bind site length on extinction time

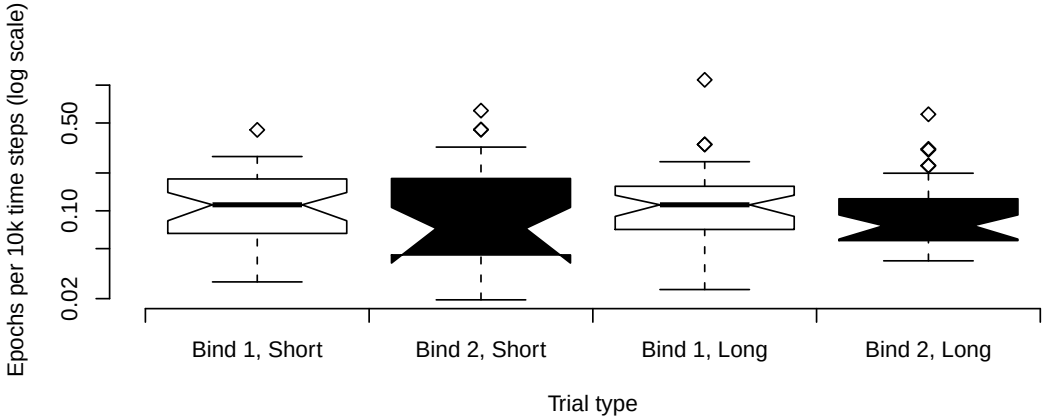


Figure 25: Effect of bind site length on the number of epochs per 10,000 time steps

D.4.2 Short vs Long bind sites

Figures 24 and 25 show the effects of the two binding schemes and the length of the bind site on the time to extinction and the number of epochs per 10^4 time steps respectively. The four trials appear similar, apart from the time to extinction for long encodings using the new bind scheme. This may be because the longer encoding allows the designed bind sites to remain dominant, whereas mutations in the middle of a shorter bind would cause molecular parings to bind elsewhere on the molecule, preventing replicase functionality.

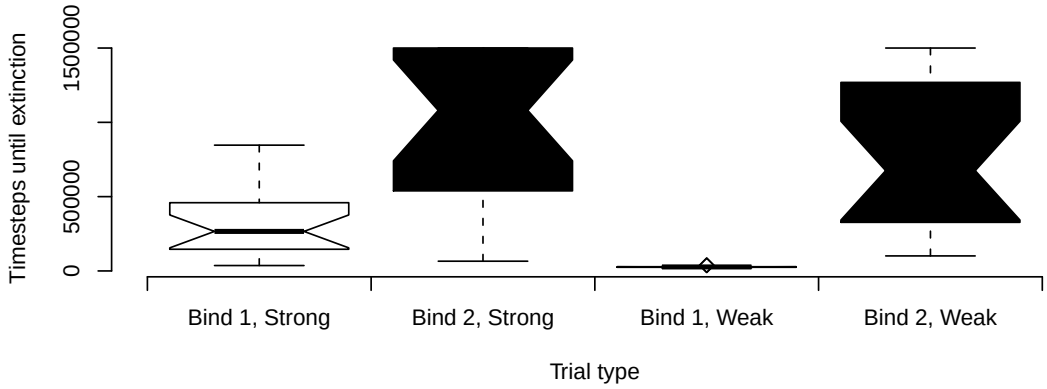


Figure 26: Effect of weak bind sites encoding on extinction time

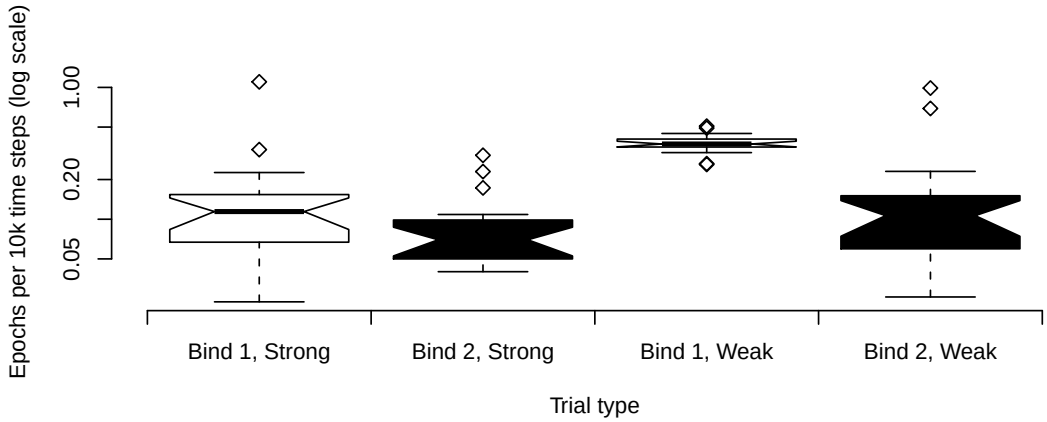


Figure 27: Effect of weak bind sites on the number of epochs per 10,000 time steps

D.4.3 Strong vs Weak bind sites

Figures 26 and 27 show the effects of the two binding schemes and the strength of the bind site on the time to extinction and the number of epochs per 10^4 time steps respectively. Here we see that the new bind scheme allows weaker binds to perform adequately. Both the strong and the weak binds show acceptable performance in terms of time and number of epochs. However, the replicase with weak binding does not self-maintain under the previous bind scheme. The high number of epochs per 10^4 time steps for this configuration is due to the fact that the weak-binding replicase goes rapidly to extinction in a single epoch that lasts for not much more than 10,000 time steps.

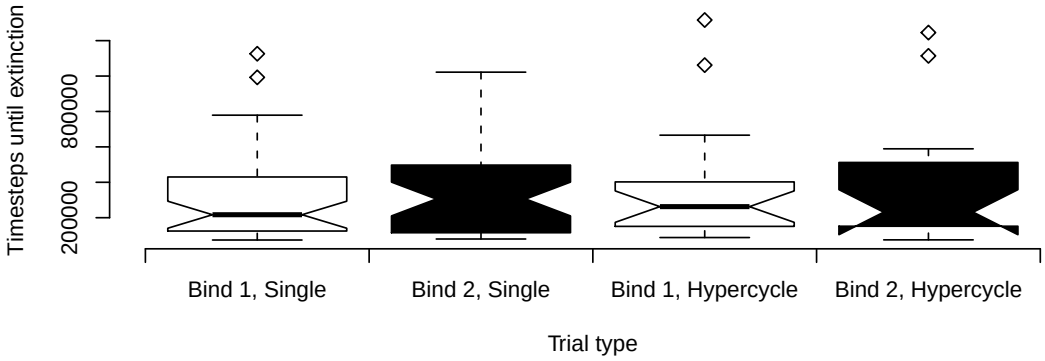


Figure 28: Effect of Hypercycles encoding on extinction time

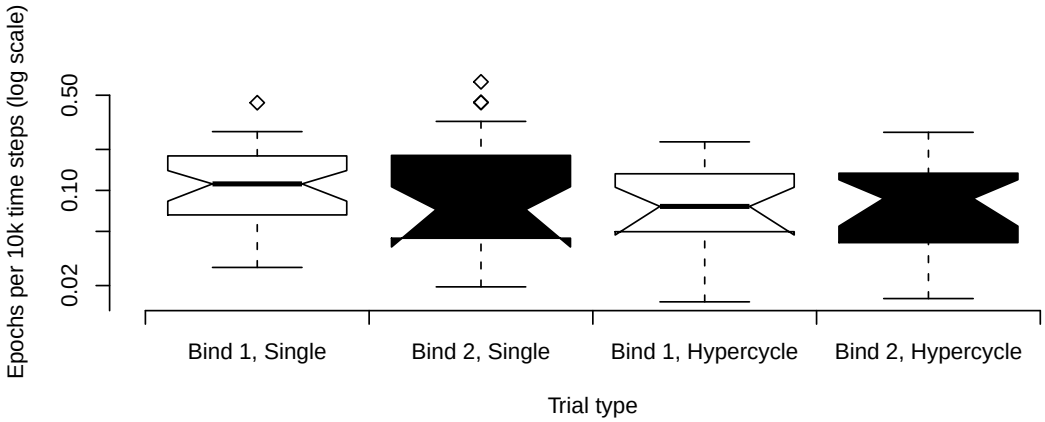


Figure 29: Effect of Hypercycles on the number of epochs per 10,000 time steps

D.4.4 Short vs Hypercyclic sites

Figures 28 and 29 show the performance of the two binding schemes for single and hypercyclic replicase systems on the time to extinction and the number of epochs per 10^4 time steps respectively. It is striking that the hypercyclic systems are broadly similar to the single replicase systems. This indicates that low-order hypercycles are not less stable than single-replicase systems, and so they can arise easily from a single replicase ancestral system.

D.5 Conclusions

In summary, we can conclude the following from the above experiments:

- The new binding scheme allows for more diversity and neutral mutations in the molecular bindings
- Encoding using ”safe” symbol sets offers some protection against parasites
- Longer bind sites tend to be more robust
- Low order hypercyclic systems are capable of competing with single-replicase systems.