Molecular microprograms

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

Departments of Computer Science[1], Electronics[2], Biology[3], University of York, UK

Abstract. Bacteria offer an evolutionary model in which rich interactions between phenotype and genotype lead to compact genomes with efficient metabolic pathways. Central to this is the expression and folding of sequences of amino acids to form proteins. We seek an analogous process that supports a rich artificial heredity. These systems can be simulated by stochastic chemistry models, but there is currently no scope for open-ended evolution of the molecular species that make up the models. Instruction-set based Artifical Life has appropriate evolutionary properties, but the individual is represented as a single executing sequence with little additional physiology. We describe a novel combination of stochastic chemistries and evolvable molecule microprograms that gives a rich evolutionary framework. Key to this approach is the use of inexact sequence matching for binding between individual molecules and for branching of molecular microprograms. We illustrate the approach by implementation of two steady-state replicase RNA analogues that demonstrate "invasion when rare".

1 Introduction

One sees elegant, evolved design solutions in all levels of life on earth, yet our artificial models of evolution seem limited by comparison. When engaging with the artificial life (ALife) community, it is easy to get the impression that we haven't got the model of genetic algorithms quite right yet, and that this is why they are typically only used as optimisers. There is a need for debate about whether the way we think about good engineering is compatible with evolutionary processes [1]. Is biology one gigantic hack, and is it the right way to do things when we think on different timescales? "Leakiness" of network pathways is a common property of biological systems, which can make the extraction of general principles difficult [2]. Attempts to tackle these issues have the potential to break new ground. Our project[1] looks at rich ways of building genotype-phenotype (geno-pheno) interactions, as this is how bacteria are known to rapidly adapt to new environments. We are in the early stages of building a bacterial genetic algorithm (GA). The emulation of biological geno-pheno coupling *requires* similar coupling between GA and ALife. Other ALife works use GAs [3], but the ALife side tends to spring complete from the GA template, which is never referred to again. We are developing a leakier approach, where the genetic template is an interactive part of the phenotype, with the goal of developing richer ALife behaviours.

We have designed a computational metabolome [4], implemented as a network of reactions between a maximum of two molecular agents; it demonstrates that even when

[1] This work is part of the Plazzmid project, funded by EPSRC grant EP/F031033/1

expressed with simple computational restrictions, it is straightforward to design a gene regulatory network. To make these networks evolvable, we need to 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. We need to match the granularity of change in the expressed proteins with the characteristics of mutation that the system possesses. Thus we find that our ALife-GA coupling *demands* 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) [5]; spatial shape-based [1]; program-based. Here we explore the program-based approach, with reference to a set of biological principles that we believe characterise biological systems.

There is a rich history of individual-as-program ALife. The key ones are Avida [6], Tierra [7] and "BF" [8]. All these place heavy emphasis on the genotype: each "organism" has a template code, plus some registers and some energy, which together confer fitness on the individual. The phenotype is forced to be simple: essentially a sequence of instructions, plus a miniature processing "factory". In these systems the factory of each organism is not subject to heritable change. We argue that the ALife organism needs to be richer. As a first step, we use executing sequences as the basis for the chemistry 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.

The term *microprogram* describes assemblies of the lowest-level instructions (*microcode*) that describes a program at the lowest possible level. By analogy, reactions at the molecule-molecule level form the microcode of the organism [9]. In our representation, molecular microprograms are implemented as sequences of instructions, loosely analogous with the way amino acids fold to form proteins. The sequences float in an abstract space, and the "program" emerges via the mixing and reacting of these molecular microprograms. In addition, by basing binding and execution pathways on inexact subsequence alignment [10], we get the benefit of fine-grained switching between execution pathways of the network that forms the high-level program.

The novelty here lies in a stochastic chemistry whose molecular species are executing microprograms, and whose binding rates and reactions are emulated via inexact subsequence alignment. As a demonstration, we have hand-written a molecular microprogram that is capable of copying other molecules that bind to it, including a copy of itself. We call this molecule a *replicase*. We show that it is feasible for mutations to change the efficacy of the microprogram via an "invasion from rare" experimental evaluation.

2 Domain model of bacterial evolution

We describe an abstract model of bacterial evolution in [11], with concepts of the accessory genome and the arrangement of plasmids and chromosomes. The emphasis there is on abstracting genome organisation, but the mechanisms for achieving and maintaining an organised genome are not detailed. Here we define a domain model ([12], as

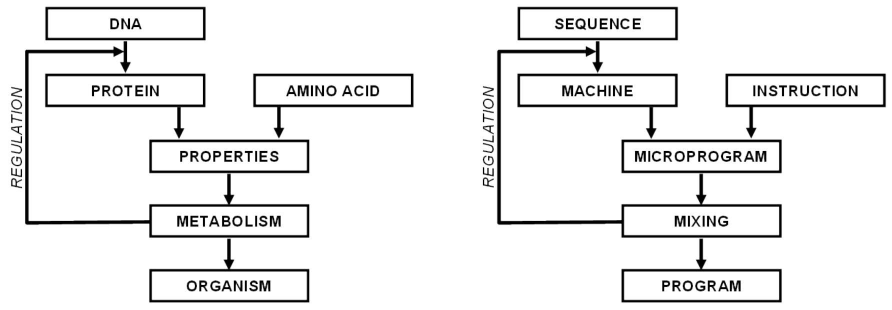

Fig. 1. Domain models of sequence-based executing machines. (a) left: a bacterial metabolism; (b) right: a computer-based simulation.

referenced in [13]) of the bacterial system we wish to emulate, and briefly discuss a microprogram-based instantiation of the resulting model.

A naive view of the genome is that it acts as an "workshop manual" for the biological entity. In bacterial systems, the genomic manual is being continually rewritten, from minor typos to new chapters to rearrangement of volumes. What is missing from most evolutionary models is the rich *machinery* for this continual re-editing process. The copying machinery is specified on the genome that it maintains, and is built the same way as other non-genomic entities in the cell.

Biological genome maintenance is carried out by proteins (folded sequences of amino acids) and sRNAs, 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. As shown in figure 1a, the functionality of the protein is 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 the shape change, which in turn changes the reaction properties. Sometimes the shape of the protein is the most significant factor: 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 timescales, the genetic code develops a deep interaction with these properties. If we wish to emulate these properties, it is imperative to get a granularity of structure that is most appropriate to the goal of open-ended heredity. Figure 1b indicates the components of a computational emulation of the biological system just described. Here a combination of the properties of low-level instructions and the sequnce they are assembled into forms the basis of a microprogram that emulates protein function. The mixing of these proteins in a stochastic chemistry allows the overall program of the system to emerge.

General Properties	Molecular Microprogram Instantiation
single, consistent, molecular representation: for any reaction A+B→C, it must be possible to describe C in the same terms as A and B	molecular microprogram: sequence of microcodes, plus single instance of four pointers
no global controller	contains information only about itself; bioentity little more than a propensity equation
structure of entity forms binding template and function	sequence of microcodes specifies template and execution
binding success proportional to some match function	Smith-Waterman alignment score as basis of bind strength
has a lifetime, eg. defined by a decay rate	decay rate is a function of sequence length
constructed as part of a sequence-reading process	undefined here; straightforward to implement
actions always local to the bound objects	molecular microprograms refer only to themselves and/or their bound partners
actions have cost, and may yield a dividend	execution of each instruction and each attempted bind costs one energy unit; dividend unimplemented: organism requires steady energy influx
actions are not absolute, but relative to the qualities of the objects they refer to	The **=** instruction inserts symbols only at the write pointer **W**: it does not overwrite anything

Table 1. Properties of an evolving artificial chemistries. General principles (left column). Properties in the molecular microprogram instantiation (right column).

3 Detailed Model

Direct imitation of biology is not currently feasible for all but the simplest systems, since computational overheads are high and biological systems are rarely understood completely. The immediate challenge is to have a specification of a phenotype that allows a rich set of properties to arise such that some template pattern can be used to build a machine. We put rich functionality into the microcodes, and use genetics to exploit their properties. Each microcode is represented by a symbol, and sequences of symbols form the microprogram of a molecule. Each molecule has a set of four pointers, which are manipulated by the microcode to carry out the function of the molecule. Microprogram execution is initiated when a molecule binds to another. When a bind occurs, the two molecules negotiate which one should act as executing program, and which should act as data. The executing molecule's pointers can point to the data molecule.

Our design is motivated by properties of biological systems that we believe to be important. Table 1 lists these general properties, and gives corresponding features of our system. For our initial investigations, we have made assumptions about what the most efficient implementation of a molecular microprogram might be. In this section, we outline the approach we are using to explore the proposals outlined above.

Our molecular microprograms are analogous to proteins and sRNAs, where sequences of amino acids fold to form molecular machines. We replace the concept of folding with program control flow for a sequence of program symbols. Our molecules

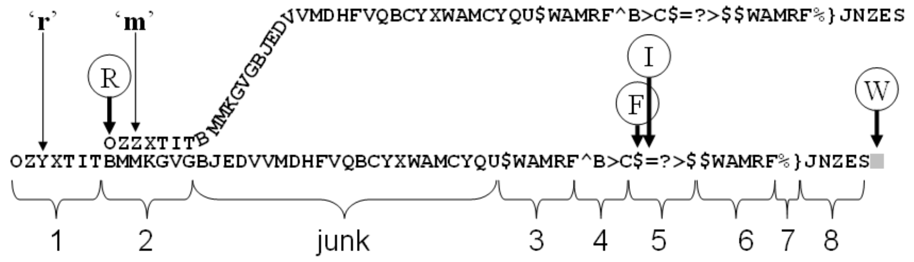
'r'
'm'
OZZXTITBMMKGVGBJEDVVMDHFVQBCYXWAMCYQU$WAMRF^B>C$=?>$$WAMRF%}JNZES
R
I
F
W
OZYXTITBMMKGVGBJEDVVMDHFVQBCYXWAMCYQU$WAMRF^B>C$=?>$$WAMRF%}JNZES
1
2
junk
3
4
5
6
7
8

Fig. 2. 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 excecuting 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.

have no shape or explicit dimension. The uniqueness of a molecule is encoded in its sequence of symbols. We place heavier emphasis on the concept of binding than do Avida and variants. In functioning microprograms, parts of the sequence describes one or more binding region and when bound, a microprogram is executed from the sequence of instructions beginning 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.

A molecule consists of a sequence of symbols, and four pointers to positions on the sequence: the instruction pointer **I**; the read pointer **R**; the write pointer **W**; and the flow pointer **F**, which is commonly used to reset the position of **I** during iterative execution. A molecule has no stacks or registers, or access to any global controllers. The symbols in the molecular sequence are instructions that manipulate the pointers, thereby implementing the molecule's function. We illustrate a replicase molecule in figure 2, which shows eight distinct regions of the microprogram. 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 as the first symbol is about to be copied: **I** indicates that the next instruction is **=** (copy). **F** is set to the beginning of region 5, which executes the iterative copy process. **R** is positioned at the start of the template molecule's sequence. **W** is positioned at the end of the executing molecule's sequence. This is where the new molecule is built.

A detailed description of our microcode implementation can be found in [14]. Table 2 provides a summary of these codes, which manipulate the pointers and control the execution pathways of the molecular microprograms.

Once created, each molecule floats unbound in the metabolism of the organism. It may bind with other molecules, in which case its microprogram may be executed. Bound pairs dissociate on termination of the microprogram. This process of binding

Code(s)	Name	Description
`A` to `Z`	`n-op`	inactive template code and instruction modifier
`$`	`search`	shift `*F` to a matching template
`>`	`move`	shift pointers to the flow pointer
`^`	`toggle`	switch pointer to molecular bind partner
`?`	`if`	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 2. Symbols and actions used in the current implementation of molecular microprograms. For a detailed description see [14].

and dissociation continues until the molecule is either destroyed whilst in a bind, or decays whilst unbound.

Binding is a complementary sequence matching process. The idea is to obtain a probability of binding based on the match. The Smith-Waterman (SW) algorithm [10] is ideal for this, since it was designed to give positive scores for similarities that are unlikely to have arisen by chance alone. When two sequences are compared, the SW algorithm finds the longest common subsequence (LCS) between them. SW calculates the tradeoff between the length and the mismatches in the LCS. A perfect match of length 5 will score 5. Each mismatch in a subsequence pairing reduces this score by some amount, and the penalty increases for mismatches that are increasingly unlikely, so a match of length 10 but with significant mismatches might score 4. Scores of zero or less indicate that the subsequence pairing is likely to have arisen by chance.

For any two molecules i and j, we use SW to detect the LCS for a sequence pair $\phi_{i,j}$, and use the score and the length to define the binding site and derive a probability of binding, $p(\phi_{i,j}) = (s/l)^l$, where s is the SW score and l is the LCS length. For details of how the mismatch penalties are calculated, see [14].

We also use $p(\phi)$ to control program flow: `$` and `?` use a template match, and have different operation if a match is not found. This "soft" execution pathway is ideal for evolutionary algorithms, since it allows incremental change in microprogram execution that mimics some of the attractive biological properties listed in table 1.

For computational simplicity, on decay a molecule is instantaneously deleted. The chance of decay is a function of the length of the sequence. This is a crude way of ensuring that things that are expensive to build tend to persist in the metabolome, without having molecule fragments floating around and reacting with other, complete molecules in the system. We use a decay probability of $1/L^2$, where L is the length of the sequence. Note that this is "passive" decay. There is scope to build a "destructase" molecule, whose microprogram would chop up anything that bound to it into smaller molecules. The resulting fragments would then be more likely to decay, since they are shorter.

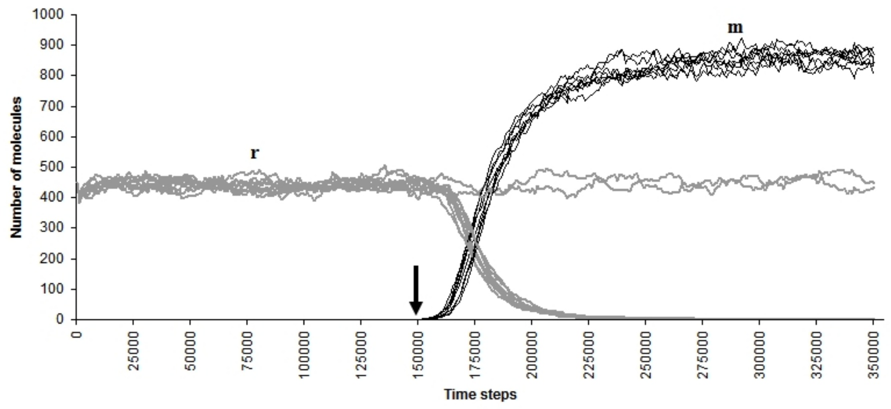

Fig. 3. Invasion when rare. Ten typical runs of a metabolism of weakly binding replicase species $\mathbf{r}$ (thick grey lines) which is invaded (indicated by the arrow) by a single molecule of mutant species $\mathbf{m}$ (thin black lines), which has a stronger binding affinity. Eight of these trial runs demonstrate "invasion when rare" by $\mathbf{m}$. In two cases, $\mathbf{m}$ went extinct, and $\mathbf{r}$ remained at equilibrium.

4 Experiments

Inspired by the RNA world model [15], 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 2. Note that there are very many species with replicase functionality in our molecular space. Regions 1 and 2 of the microprogram for $\mathbf{r}$ are complementary sequences of match length $l = 7$, match score $s = 5.875$ and $p(\phi_{\mathbf{r},\mathbf{r}}) = 0.293$. 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 $l = 7$, $s = 7$, $p(\phi_{\mathbf{m},\mathbf{m}}) = 1$, and importantly $p(\phi_{\mathbf{r},\mathbf{m}}) = 1$ also: $\mathbf{r}$ binds to $\mathbf{m}$ more readily than $\mathbf{r}$ binds to $\mathbf{r}$.

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 3. 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.

5 Conclusion

We have demonstrated that an ALife environment that uses a sequence of instructions as the basis for interaction and program execution can show important biological properties. We intend to use this platform to explore these ideas further, in particular to develop a fully functioning bacterial emulator.

Our experiment in this paper used a "hand selected" mutation that was guaranteed to give beneficial effects. The experiment demonstrated that a single instance of a beneficial mutation can easily become the dominant species in our system. The idea now is to see how different mutation rates affect a functioning molecular species. Mutation can be implemented straightforwardly by allowing the **`copy`** operator to introduce changes to the copy. It may be possible to determine the "error catastrophe" levels for this system, and use this limit to derive appropriate mutation rates.

References

1. Bentley, P.J.: Fractal proteins. Genetic Programming and Evolvable Machines (March 2004) 71–101
2. Kitano, H.: Systems biology: a brief overview. Science **295**(5560) (March 2002) 1662–1664
3. Knibbe, C., Fayard, J.M., Beslon, G.: The topology of the protein network influences the dynamics of gene order: From systems biology to a systemic understanding of evolution. Artif. Life **14**(1) (2008) 149–156
4. Hickinbotham, S., Clark, E., Stepney, S., Clarke, T., Young, P.: Gene regulation in a particle metabolome. In: CEC 2009. (2009) in press
5. 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)
6. Pennock, R.T.: Models, simulations, instantiations, and evidence: the case of digital evolution. J. Exp. Theor. Artif. Intell. **19**(1) (2007) 29–42
7. Ray, T., Xu, C.: Measures of evolvability in tierra. Artificial Life and Robotics **5**(4) (December 2001) 211–214
8. 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
9. Danchin, A.: Bacteria as computers making computers. FEMS Microbiology Reviews **33**(1) (2009) 3–26
10. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. J Mol Biol **147**(1) (1981) 195–197
11. Stepney, S., Clarke, T., Young, P.: Plazzmid: An evolutionary agent-based architecture inspired by bacteria and bees. In: ECAL 2007. (2007) 1151–1160
12. Andrews, P.S., Sampson, A.T., Polack, F., Stepney, S., Timmis., J.: CoSMoS development lifecycle, version 0. Technical Report (in preparation), University of York (2008)
13. Garnett, P., Stepney, S., Leyser, O.: Towards an executable model of auxin transport canalisation. In Stepney, S., Polack, F., Welch, P., eds.: Proceedings of the 2008 Workshop on Complex Systems Modelling and Simulation, Luniver Press (2008) 63–91
14. Hickinbotham, S., Clark, E., Nellis, A., Pay, M., Stepney, S., Clarke, T., Young, P.: An abstract metabolism of molecular microprograms. Technical Report (in preparation), University of York (June 2009)
15. Lincoln, T.A., Joyce, G.F.: Self-sustained replication of an RNA enzyme. Science (January 2009) 1167856+