

Investigating self-fabrication in the context of artificial chemistries

by

Christopher van Niekerk

*Thesis presented in partial fulfilment of the requirements
for the degree of Master of Science (Biochemistry) in the
Faculty of Science at Stellenbosch University*



Department of Biochemistry
University of Stellenbosch
Private Bag X1, 7602 Matieland, South Africa

Supervisor: Prof. J.-H.S. Hofmeyr (supervisor)

December 2014

Acknowledgements

At the end of my Honours year, someone said to me: ‘So, [do] you think you can complete a Masters?’. With a rye smile, and a slight tinge of arrogance, I replied with a simple ‘Yes’. Of course, at the time, I brushed it off as something easily accomplished, how could that person think that *I* would not be able to complete a simple Masters? After all, in an academic environment, everyone just seems to be completing different degrees all the time.

What was said to me was not condescending, and now, standing at the end of this journey, I look back and feel that what was asked was actually ‘Do you have it inside you to persevere, no matter how great or small the challenge?’. And now, still with my rye little smile, but with arrogance beaten off my face, I can reply with a simple ‘No’.

No, I do not have what it takes to always persevere, not without the constant help and support of my mentors, role models, family and friends. For all their help, support, love, and encouragement I would like to extend a heartfelt thank you. Thank you for not giving up on me, even when I had given up on myself. I only hope that one day I might be able to help others, as all of you have helped and nurtured me.

I would like to extend a special thank you to Professor Jannie Hofmeyr for spurring my interest in computational systems biology, and for being

DECLARATION

an outstanding supervisor throughout my Honours and Masters years. I would also like to thank Professors Johann Rohwer and Jacky Snoep for their advice and support. Thank you also to Dr Danie Palm for many of the interesting ideas and conversations concerning my work.

I would also like to thank my friends, family members, and loving fiancé, for all their kindness, love, and constant support throughout my endeavours. Without all of you, I would not be standing where I am today.

Thank you.

Contents

Declaration	i
Contents	iv
List of Figures	vi
List of Tables	viii
Summary	ix
Opsomming	xi
1 Introduction	1
2 Artificial Chemistries	4
2.1 What are artificial chemistries?	5
2.2 Types of artificial chemistries	6
2.3 Formal frameworks for artificial chemistries	9
2.3.1 Simple Arithmetic Operators	9
2.3.2 Matrix Multiplication	11
2.3.3 Binary String Automata	12
2.3.4 Polymer Chemistry	17
2.3.5 Typogenetics	20

2.3.6	Explicit reaction networks: Explicit spatial reaction system	23
2.3.7	Lambda Calculus	25
2.4	Application of artificial chemistries	29
3	Binary String Systems	30
3.1	Binary Sequences	30
3.2	Sequence Folding: Mapping	33
3.3	Binary String Interactions	36
3.4	Model design	38
3.4.1	Population dynamics	39
3.4.2	Lethal strings	40
3.4.3	Model comparison	41
4	Results	44
4.1	Reaction tables	44
4.2	Simple Homogeneous Systems	55
4.3	Random and Selective Homogeneous Systems	82
4.4	Heterogeneous Systems	109
5	Discussion	123
5.1	Critique	138
5.2	Future research	142
	Bibliography	144
	Appendix	150
	Program listing	150

List of Figures

2.1	Number division chemistry	10
2.2	Binary string automata	14
2.3	Binary string automata: Early evolutionary behaviour	16
2.4	Binary string automata: Long-term evolutionary behaviour	17
2.5	Autocatalytic polymer chemistry	19
2.6	Typogenetics: The GC strand	22
2.7	Explicit reaction networks: Initial 7 instants	25
2.8	Explicit reaction networks: Instants 44-47	26
4.1	Homogeneous systems: S_7 canonical folding	57
4.2	Homogeneous systems: S_9 transpose topological folding	62
4.3	Homogeneous systems: S_6 canonical folding	65
4.4	Homogeneous systems: S_{11} topological folding	68
4.5	Homogeneous systems: S_9 topological folding A	72
4.6	Homogeneous systems: S_9 topological folding B	72
4.7	Homogeneous systems: S_6 transpose topological folding	76
4.8	Homogeneous systems: S_6 topological folding	79
4.9	Homogeneous systems: S_6 random folding	83
4.10	Homogeneous systems: S_6 selective folding	91
4.11	Homogeneous systems: S_6 random folding, zoomed	98

4.12 Homogeneous systems: S_9 random folding	100
4.13 Homogeneous systems: S_9 selective folding	104
4.14 Heterogeneous systems: Destructor decay (1-2)	110
4.15 Heterogeneous systems: Destructor decay (3-4)	111
4.16 Heterogeneous systems: Destructor decay (5-6)	112
4.17 Heterogeneous systems: Destructor & Exploiter decay (1-2) . .	114
4.18 Heterogeneous systems: Destructor & Exploiter decay (3-4) . .	114
4.19 Heterogeneous systems: Destructor & Exploiter decay (5-6) . .	115
4.20 Heterogeneous systems: Substitution $N=100$ (1-2)	117
4.21 Heterogeneous systems: Substitution $N=100$ (3-4)	117
4.22 Heterogeneous systems: Substitution $N=100$ (5-6)	118
4.23 Heterogeneous systems: Substitution $N=1$ (1-2)	120
4.24 Heterogeneous systems: Substitution $N=1$ (3-4)	121
4.25 Heterogeneous systems: Substitution $N=1$ (5-6)	122

List of Tables

2.1	Examples of λ -expressions with syntactic similarity.	28
4.1	Number of types of reactions per folding type	44
4.2	String production frequency for all reaction tables	46
4.3	Reaction table: Canonical folding	51
4.4	Reaction table: Transpose canonical folding	52
4.5	Reaction table: Topological folding	53
4.6	Reaction table: Transpose topological folding	54
4.7	String production for new string generating reactions	55

Summary

This thesis gives a broad overview of what artificial chemistries (ACs) are, a brief review of several ACs and their applications, and an in depth analysis of one specific AC: the four-bit binary string system. The model designed by Banzhaf [1] for *in silico* examination was recreated using the Python programming language. The initial motivation was to identify an existing AC that could be used to elucidate the sequence-function relationship, which led to the simultaneous investigation of self-organization in AC systems [7]. The interest in sequence-function relationships stems from their importance for self-production of objects [35]. For self-replication to be possible in larger organizations, the components of the organization must be able to continuously produce themselves [3, 7]. We chose the four-bit binary string system for investigation because of its simple design and implementation, its ability to yield complex results from interactions between a small population of objects, and its analogy to the DNA–RNA–protein organisation. When a population of objects are allowed to continuously interact, self-production and self-organization occur, even in simple artificial systems [7, 8]. The stability of the emergent organizations depends on the interactions of its components, which must be capable of self-production if they are to maintain the organization [27]. Self-production of objects depends on their sequence-function relationship, which determines their rate

of replication when interacting with other objects.

Opsomming

Hierdie tesis verskaf ‘n brëe oorsig van die algemene aard van artifiële chemies (ACs), ‘n kort opsomming van ‘n paar ACs en hul toepassings, en ‘n diepgaande analise van een spesifieke AC: die 4-bis binêre stringstelsel. Die model wat Banzhaf [1] ontwerp het vir *in silico* eksperimentering is hier herskep in die Python programmeringstaal. Die aanvanklike motivering was om ‘n bestaande AC te identifiseer wat gebruik kon word om die sekvens-funksie verwantskap te ontrafel, en dit het gelei tot die gelyktydige ondersoek van self-organisasie in AC stelsels [7]. Ons belangstelling in sekvens-funksie verwantskappe spruit uit hul belang vir die selfproduksie van objekte [35]. Om selfreplisering in meer omvangryke organisasies moontlik te maak moet die komponente in staat wees om hulself eenstryk te produseer [3, 7]. Ons het ‘n 4-bis stelsel vir hierdie studie gekies omdat die ontwerp en implementering eenvoudig is, omdat interaksies binne ‘n klein populasie van objekte komplekse resultate gee, en omdat die stelsel se organisasie analoog aan die DNA-RNA-proteïen organisasie is. Wanneer ‘n populasie van objekte toegelaat word om eenstryk op mekaar te reageer vind self-produksie en self-organisasie vanself plaas, selfs in eenvoudige artifiële stelsels [7, 8]. Die stabiliteit van die emergente organisasies hang af van die interaksies tussen die komponente, wat self die vermoë tot selfproduksie moet hê indien hulle die organisasie in stand wil hou [27]. Selfproduksie

van objekte hang af van hul sekwens-funsieverwantskap, wat op hul beurt bepaal hoe vinnig hulle repliseer wanneer in interaksie met ander objekte.

Chapter 1

Introduction

Self-fabrication is one of the main features that distinguishes living from non-living systems [30]. Growth, self-maintenance, adaptation and reproduction all depend on self-fabrication [2]. Hofmeyr [30] showed that two processes that occur in all living systems, namely polypeptide and polynucleotide folding and unassisted self-assembly of molecular machines such as ribosomes, spliceosomes, proteasomes and chaperones, are essential for self-fabrication. The sequence information in DNA is transcribed and translated into the sequence information in polypeptides, but in order for the polypeptides to assume their cellular function they have to fold into three-dimensional conformations [4, 9]. Molecular machines are made up of numerous components that have to be assembled into whole functional structures. If these processes required another molecular machine then the assembly of this machine would need another machine, which would need another machine, leading to an infinite regress. The ability to self-assemble without any help from other structures circumvents this problem. The ‘information’ on how to assemble a multi-component molecular assembly is embedded in the components themselves. The living cell is therefore

a closed system in terms of efficient causation [44]. The complexity of self-fabrication is thus to be found in the way the cellular processes are functionally organised [30, 38, 50].

The initial motivation for the work described in this thesis was to identify an existing AC that could be used to elucidate the sequence–function relationship. Specifically, we wanted to identify a system analogous to the DNA–RNA–protein system, with objects that can exist in ‘sequential’ and ‘functional’ states [4]. The interest in sequence–function relationship arises from its importance in self-producing objects, which are crucial components of self-replicating systems [27]. The initial investigation of the sequence–function relationship lead to the simultaneous investigation of self-organization in AC systems.

Chapter 2 gives a brief review of ACs and their current applications. The objective for this section was to give an overview of what ACs are, how they are designed, how they differ, and what they are used for in practice. This section also describes several ACs that were initially investigated, and why we eventually chose to investigate binary string systems.

Chapter 3 provides a detailed analysis of the the four-bit binary string system, which was chosen over other ACs and matrix multiplication systems because it is the AC with the smallest non-trivial string lengths, and because of its ability to display behaviour similar to larger string systems while retaining simple design and implementation.

Chapter 4 presents the results of the investigation of the behaviour of the model described in Section 3.4. Section 4.1 reviews reaction tables and their contents, and describes the different types of operators and strings that can constitute populations. Section 4.2 examines simple homogeneous systems, which were initialised with only one type of string. Section 4.3 further

examines systems homogeneous systems, but using random and selective folding instead of standard folding. Section 4.4 examines 24 scenarios of heterogeneous systems. Four groups of six systems were investigated, each group using a different type of substitution method.

Chapter 5 presents a discussion and detailed analysis of the systems and organizations described in Chapter 4 by considering the internal relationships between system components. The relationships between groups of objects in the systems are examined, and their role in the emergence of the resulting organizations is described.

Chapter 2

Artificial Chemistries

The theory of Artificial Life (AL) was theorized before computers as we know them existed, and initial computers were unable to implement AL systems, simply because they were not designed to. Alan Turing had other ideas, and he was one of the first people to integrate AL logic programs with computer systems [45]. Since his time almost every model of computer has been used to implement an AL system in some way or another [4, 11, 13, 15]. The term ‘artificial chemistry’ only appeared in the field of AL in the 1990’s [43], but models that would later fall under the definition of an AC appeared much earlier [49]. Today still, every generation of computer utilizes, and is also constricted by, the infrastructure of the hardware available.

In this project we examined aspects of self-fabrication using an AC based on a self-organising system of binary strings. This particular system was developed in the 1990s by Banzhaf [1, 2]. Their system consisted of sequences of binary numbers that came in two forms, a 1-dimensional ‘sequence’ form and a 2-dimensional ‘catalytic/operational’ matrix form. The sequence form is operated on by the operational form, which is generated by spontaneous ‘folding’ up of the sequence form. A system of elements

that react in such a manner is capable of network formation. These networks may be auto-catalytic in nature, and so display emergent artificial self-fabricating systems [1–3, 11, 14, 37]. In this project ACs were used to study examples of self-fabricating networks, but what exactly are ACs, how are they made, and why are they useful?

2.1 What are artificial chemistries?

In short, an AC is a man-made system that is similar, in some aspects, to a real chemical system: it is a set of rules stating which reactions occur, and with what strength [19, 24, 26, 32]. There are many different types of ACs that are differentiated by their design, with each type of AC designed to try and answer a different question. One could say that when we create ACs we are trying to create appropriate computational and mathematical tools for describing, studying and understanding complex chemical systems, particularly aspects of structure formation, self-organization, chemical information processing and pre-biotic molecular evolution [18, 19, 24, 32].

ACs are more often described as a system composed of three main parts:

1. A set of objects or molecules, S : objects may be numbers, abstract symbols, character sequences, binary strings, lambda-expressions, proofs or abstract data structures [16, 19, 24, 26].
2. A set of reaction rules, R , that describe the interactions among objects: the reaction rules depend on the types of objects in the system and may be simple arithmetic operations, finite-state machines, Turing machines, string matching, string concatenation, matrix multiplication, lambda calculus, proof theory, boolean networks or cellular automata [16, 19, 24, 26].

3. An algorithm, A , that drives the system and defines the population dynamics: the algorithm describes how the reaction rules are applied to the set of objects or molecules, and it may simulate different reactions, including ordinary differential equations, explicit collision simulations, well stirred reaction vessels with no topology, self-organizing topologies, continuous 3D space or fixed 3D Euclidean space [16, 19, 24, 26].

These three components are the cause of ACs usually being defined in terms of a triple (S, R, A) : S representing the objects or molecules, R describing the collision rules that govern molecular interaction, and A representing the algorithm driving the reactions forward [6, 19, 26].

2.2 Types of artificial chemistries

Although almost all ACs can be defined in terms of a triple, each AC is still unique. This is because each AC is designed to try and solve a specific problem or answer a specific chemical question. This causes ACs to be differentiated by their components along three main axis:

1. Analogous vs. abstract models.
2. Constructive vs. non-constructive systems.
3. Explicit vs. implicit objects and reaction rules.

Levels of abstraction vary between ACs based on how analogous or abstract they are: in analogous ACs there is a relation between each molecule and reaction in the AC to molecules and reactions in a real chemistry; in abstract ACs there may be little or no relation to molecules or reactions in real chemistries while there are similarities on the macroscopic level [19, 22, 24].

2.2. Types of artificial chemistries

Constructive systems differ from non-constructive systems in that they are capable of producing new objects in addition to objects the system was initialised with, unlike non-constructive systems where the objects given explicitly in the beginning of the model are the only objects available to the system [24, 26]. This allows constructive systems to develop new dynamic states, as opposed to non-constructive systems where the system dynamics are limited by the initial objects. Constructive systems can also be divided into *weakly constructive* and *strongly constructive* systems. In systems that are weakly constructive new components may be randomly generated through different mechanisms. This method, albeit *explicit* and imposed, seeks to simulate natural occurrences such as random mutation, which, in nature, may be caused by a variety of sources, including cosmic radiation. Strongly constructive systems are capable of generating new objects through the interaction of existing objects, and not through random generation. Constructive systems potentially allow the construction of an infinite number of different substances, all facilitated through the *implicit* reaction mechanism, which relies on the structure of the interacting objects [12, 19, 23, 24].

A system is very rarely only implicit or explicit as different parts of the system may be either. A reaction rule is implicit if it uses structural aspects of the objects, and implicit reaction rules can also govern explicit populations of structured objects (populations in which no new objects can appear). Populations of objects (S) and reaction rules (R) that govern object interaction can be explicit or implicit, while algorithms (A) can only be explicit.

Objects (S): Systems where objects do not have defined structure contain explicitly defined populations, and these systems do not allow for constructive processes or the study of the structure-function relationships. Objects in such systems are usually abstract symbols. Implicitly designed objects are constructed from smaller components, and the resulting representation of the object is its structure. Objects with different structural components also allow for implicit reaction mechanisms that rely on the structure of the components [19, 23, 24, 26].

Reaction rules (R): Implicit reaction rules use different structural components of objects to generate new objects. Explicit reaction rules are arbitrarily defined by the user and they act on structureless objects, e.g., abstract symbols. Population size can also be implicitly or explicitly regulated, depending on the reaction mechanism. If the population is kept constant by continuous object consumption and production then the reaction mechanism is implicit, while the mechanism is explicit if the number of objects in the population is fixed [19, 24, 26].

Algorithms (A): Algorithms will always be explicit as the definition of population dynamics must be defined from outside of the system. This inability of artificial objects to spontaneously react is one of the challenges facing ACs: as of yet it has not been possible to create a spontaneously reacting system; system time must be created through an imposed algorithm, which will always be explicit [19, 24, 26].

2.3 Formal frameworks for artificial chemistries

This section gives a brief overview of some existing frameworks used to construct ACs and an example of the each type of system.

2.3.1 Simple Arithmetic Operators

The first example, number division chemistry, shows how relatively simple arithmetic operator systems can be quite constructive despite their simplicity [19, 22, 26].

Objects (S): The molecules in number division chemistry are natural numbers greater than one: $S = \{2, 3, 4, \dots\}$. The size of a population is fixed at a magnitude of two or larger [19].

Reaction rules (R): Normal mathematical division is used to calculate the reaction products. Reactions take place if s_1 is divisible by the s_2 without remainder. s_1 is then transformed into the s_1/s_2 , with s_2 acting as a catalyst in the reaction. The reaction can be implicitly defined as:

$$R = \{s_1 + s_2 \rightarrow s_3 + s_2 : s_1, s_2, s_3 \in S \wedge s_3 = s_1/s_2 \text{ where } s_1 \bmod s_2 = 0\}$$

There is no reaction rule for objects that react and produce a value with a remainder, consequently these reactions do not take place and are considered elastic reactions, i.e., if $s_1 \bmod s_2 \neq 0$, then $s_1 + s_2 \rightarrow s_1 + s_2$ [19, 22].

Algorithm (A): Random collisions are explicitly simulated by selection of two random population members for interaction [19, 22].

Fig. 2.1 shows the dynamic behaviour of a system using this algorithm.

2.3. Formal frameworks for artificial chemistries

This system was initialised with population $M = 100$ with objects of the population in the range of $S \in \{2, 3, \dots, 10000\}$. The diversity of the population (measured as the fraction of primes in the population) starts very high (almost 1.0), while the prime number concentration begins below 0.1. The prime number concentration increases steadily from 50 to 150 generations before plateauing for 50 generations. At 200 generations the prime number concentration begins a stage of steep growth, while the diversity of the population begins a phase of accelerated decline. At 250 generations the diversity reaches a transient minimum, and at the same time the prime number concentration experiences a decelerated growth rate. This continues until just after 350 generations, where the system is filled with prime numbers and the diversity of the system has reached the minimum of 66% [26].

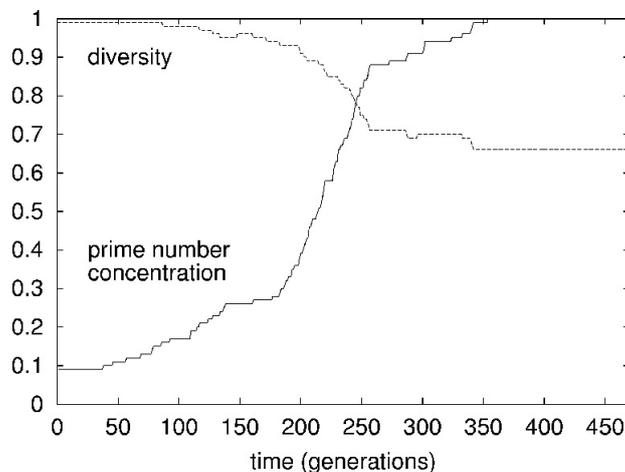


Figure 2.1: The first 450 generations of a number-division chemistry simulation. The population was initialized with $M = 100$ random numbers $S \in \{2, 3, \dots, 10000\}$ [26].

The diversity of the population remains constant by dividing larger numbers into smaller numbers, but eventually all numbers have been divided

 2.3. Formal frameworks for artificial chemistries

into their prime factors, and the diversity now decreases as concentration of the prime factors increases. Eventually all numbers are prime factors of the initial population [17, 19, 22, 26].

2.3.2 Matrix Multiplication

Matrix multiplication is discussed in detail in chapter 3, and only a brief overview will be given here. The central idea is to fold a string into a matrix, which will operate on another string by multiplication [1, 2, 4].

Objects (S): The objects are binary strings, $S \in \{0, 1\}^*$, $*$ denoting the length of the strings [1]. Experiments have been performed with fixed length (4-bit, 9-bit and 16-bit) and variable length strings [5]. Here we examine $S \in \{0, 1\}^4$.

Reaction rules (R): Assume the reaction $S_1 + S_2 \implies S_3$. The general approach occurs in three steps:

1. Fold a randomly selected string into a matrix M , according to a folding F [2, 3]. Example $S_1 = (s_1^1 \cdot s_1^2 \cdot s_1^3 \cdot s_1^4)$

$$F : M_1 \rightarrow \begin{bmatrix} s_1^1 & s_1^2 \\ s_1^3 & s_1^4 \end{bmatrix}$$

2. M is then multiplied with subsequences of S_2 : let $S_2 = (s_2^1 \cdot s_2^2 \cdot s_2^3 \cdot s_2^4)$ be divided into subsequences $S_2^{12} = (s_2^1 \cdot s_2^2)$ and $S_2^{34} = (s_2^3 \cdot s_2^4)$. M is then multiplied with S_2^{12} and S_2^{34} to form S_3^{12} and S_3^{34} : $S_3^{12} = M \odot S_2^{12}$ and $S_3^{34} = M \odot S_2^{34}$. Threshold multiplication maps the result vector to a binary vector, ensuring that values remain bit elements [2, 3].

3. The string S_3 is composed by concatenating the S_3 subsequences: $S_3 = S_3^{12} \oplus S_3^{34}$. This forms the new string $S_3 = (s_3^1 \cdot s_3^2 \cdot s_3^3 \cdot s_3^4)$ [2, 3].

2.3. Formal frameworks for artificial chemistries

Algorithm (A): The general algorithm for matrix multiplication simulates random object collision by randomly select two strings and applying the reaction rule. The population can be allowed to continuously grow or can be kept constant by the removing a random string from the population [2, 3].

These systems usually result in the formation of a steady state, where core members support each other in production. These core groups form stable auto-catalytic cycles, and groups that do not lock into beneficial reaction cycles disappear due to competition in the system [3, 6].

In later work the reactor was modelled as a topology, allowing spacial interaction between reactants [8]. Interestingly, string species that would die out in unbound reaction space due to competition could survive in the surrounding areas of other string groups, resulting in the formation of membrane-like structures [8].

Smaller systems can also be modelled with differential rate equations that accurately model the reactions in the simulation, and even systems with small objects, e.g. 4-bit binary strings, show surprising complexity in the networks they create [1–4, 6]. This will be discussed extensively in Chapters 3–4.

2.3.3 Binary String Automata

This system was inspired by biochemical reactions in the RNA world, where molecules, which are in different states, interact to produce new molecules [6, 8]. The automata reaction is based on a formal finite state automaton, which is a mixture of a Turing machine and a command driven register machine [20, 39, 47]. This system was inspired by typogenetics and tries to extract the logic underlying information processing as opposed to simulating

 2.3. Formal frameworks for artificial chemistries

physical details [20, 26, 34].

Objects (S): The objects are binary strings of a fixed length, 32-bit binary strings: $S \in \{0, 1\}^{32}$.

Reaction rules (R): A reaction takes the form of a random collision in which two random strings are selected to interact and produce a third string. The rules are second-order catalytic reactions of the form $S_1 + S_2 + X \longrightarrow S_1 + S_2 + S_3$, which can be shortened to $S_1 + S_2 \Longrightarrow S_3$. The interaction of the objects is completed in two steps: (1) S_1 is mapped to A_{S_1} , a state automaton, by interpretation of S_1 as 4-bit machine code. (2) The automaton, A_{S_1} , is then applied to S_2 to produce S_3 . After the reaction the automaton is discarded but S_1 is unchanged [26, 47].

Algorithm (A): The reactor algorithm selects two objects from the population, $M = \{S_1, \dots, S_M\}$, $S \in \{0, 1\}^{32}$. The reaction takes place if $S_1 + S_2 \Longrightarrow S_3$ is allowed, a randomly selected object is then removed to compensate for the addition. The filter $f : S \times S \times S \rightarrow \{true, false\}$ is used to easily introduce elastic reactions. Filter conditions often used in binary automata include the inhibition of operator replication and disallowing interacting strings to be the same: $f_1(S_1, S_2, S_3) = (S_1 \neq S_2 \wedge S_1 \neq S_3)$ [26, 39, 47].

Three types of macroscopic measurements were designed for evaluating binary automata: (1) diversity was taken as the number of different strings in the population divided by the total size of the population. (2) the productivity was defined as the probability of a reaction meeting the filter requirements. (3) the innovation value of a system is the probability of a reaction producing a string that has never been seen in the system before [39, 47].

Fig. 2.2 shows the structure of an automaton, containing two 32-bit

2.3. Formal frameworks for artificial chemistries

registers, the operator register and the IO register. S_2 is read into the IO register. S_1 is read into the operator register where it is mapped into a series of 4-bit sequences [21, 47]. Fig. 2.2 also contains two code tables that relate

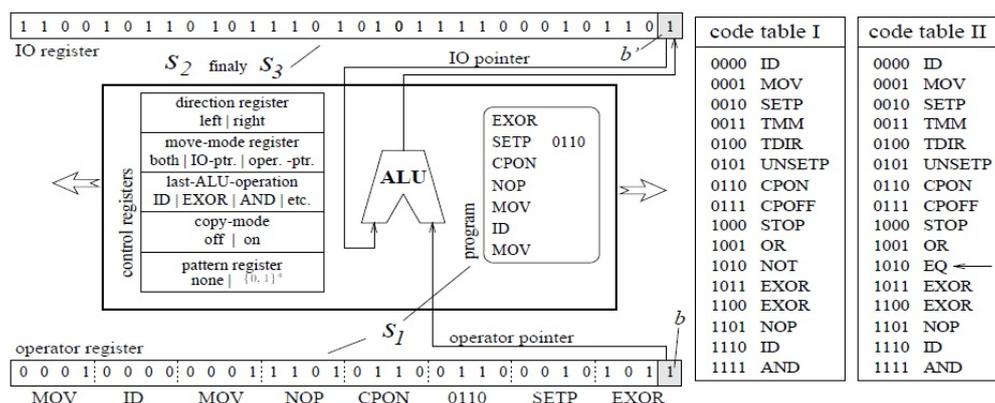


Figure 2.2: A 32-bit binary string automaton performs the reaction $S_1 + S_2 \implies S_3$. S_1 is read into the operator register and mapped into an instruction sequence. S_2 is read into the IO register. S_3 is produced by overwriting S_2 using the instruction set [20, 21].

the 4-bit sequences to corresponding instructions [25]. The instructions are then executed sequentially, starting with the first instruction and ending after the last instruction has been executed. This instruction set did not contain control statements for jumps or for looping. Each register also has a pointer that refers to a bit location. Pointers are always initialised at position 0 as shown in Fig. 2.2. The IO pointer refers to position b' in the IO register, and the operator pointer refers to the position of b in the operator register. Bits b and b' are used as inputs for the arithmetic logic unit (ALU), which performs logic operations on the two bits. The result of the ALU operation is stored in the IO pointer location and replaces the b' value [21, 39, 47].

2.3. Formal frameworks for artificial chemistries

The execution of logic operations (ID, AND, OR, EXOR, EQ, or NOT) is divided into two steps: (1) in the first step the ALU computes the product of b and b' and stores the output of the logic operation in b' . (2) in the second step the pointers are moved according to the direction register, TDIR. The move mode register, toggled by TMM, determines which pointers (operator, IO or both) are moved. MOV is the most complicated instruction as it depends on all the control registers. If copy-mode is off and the pattern register is empty then the pointers are moved according to the direction and move-mode registers. If there is a pattern (4-bit string) loaded into the pattern register the pointers are moved (max 32 steps) until the pattern is found in the operator register. If the copy mode is switched on then the last ALU operation is executed before each step. CPON and CPOFF switch copy mode on and off respectively. This allows one MOV instruction to execute many ALU operations. The pattern register is set by the 8-bit command, SETP (0010), which includes the immediate next four 4-bits after 0010 as part of the command. The pattern register is set to 'none' by the command UNSETP. The NOP command is for no operation, and the automaton will move to the next instruction upon encountering NOP in an instruction set. The system will halt upon reading the STOP command even if it has not yet reached the end of the instruction list [21, 39, 47].

Fig. 2.3 shows the first 300 generations of a binary string automaton. During this stage the evolutionary process is taking place on the binary string level, where competition is mainly between individual entities as opposed to core groups, and the system generates completely new strings at a much higher frequency than later on [21].

Fig. 2.4 shows the first 7000 generations of a different binary string automaton with a randomized population of 10^5 strings. After the initial

2.3. Formal frameworks for artificial chemistries

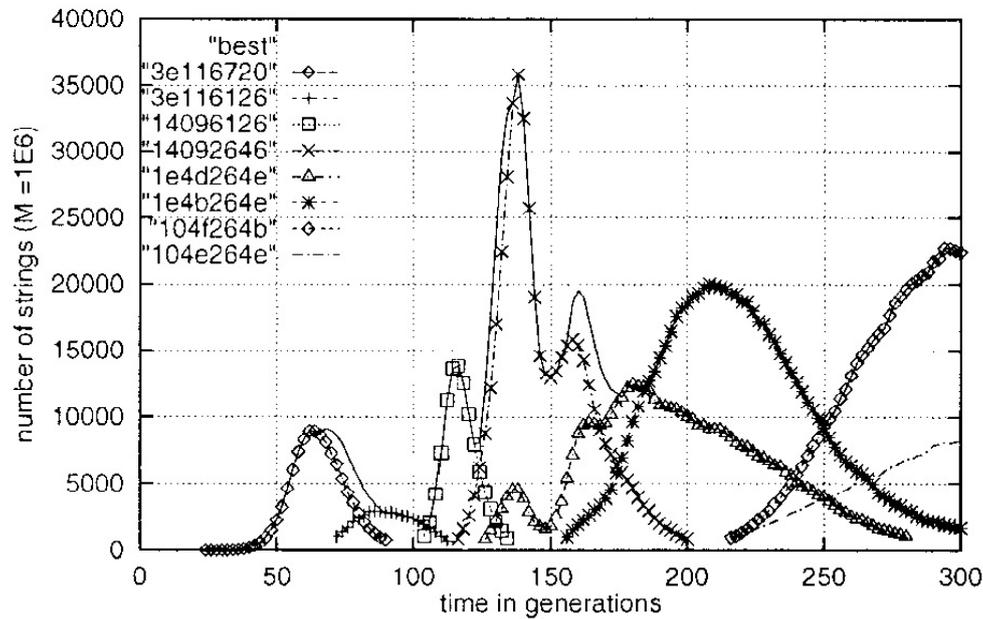


Figure 2.3: Early evolutionary behaviour in a 32-bit string automaton. A few string concentrations representative of the population are shown. The population was seeded with $M = 10^6$ random strings. Code table II from Fig. 2.2 and filter condition f were used [21].

exploratory phase, generations 0–110, and the early evolutionary phase, generations 110–200, an organization comprising approximately 50000 different string types has emerged. Only around 3000 string types of the organization are present at one time, and the organization continues to produce entirely new strings, but at a lower rate than the exploratory phase. By generation 200 the system has reached a state where there is a high probability of a reaction meeting the filter requirement (see the bottom graph of productivity (Prod) versus time in Fig. 2.4). This property of the emergent organization is a direct result of selection against active replicators and self-reactions (which will be discussed in Chapter 3) [21, 34, 47].

The productivity between generations 500–7000 shows how this core organization is still capable of further development. During this period

2.3. Formal frameworks for artificial chemistries

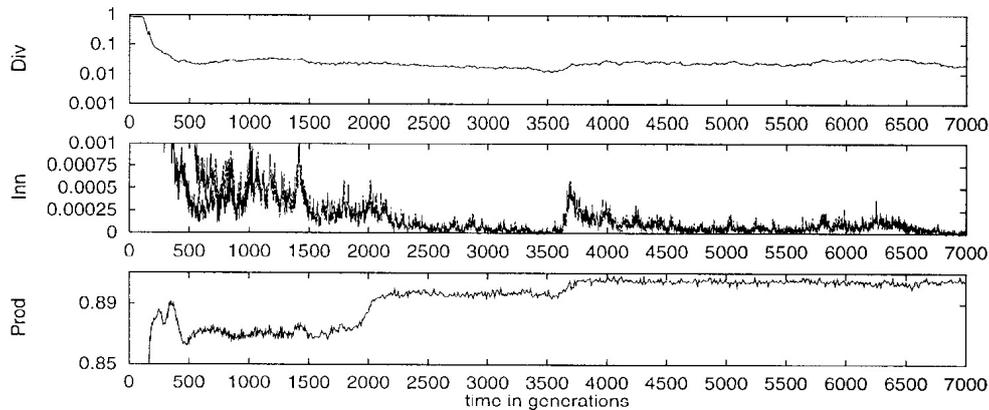


Figure 2.4: Long-term evolutionary behaviour in a 32-bit string automaton showing the diversity, innovation and productivity of a system. The population was seeded with $M = 10^5$ random strings. Code table II from Fig. 2.2 and filter condition f were used [21].

evolution is taking place on an organizational level as opposed to evolution on string level, Fig. 2.3. The core set of strings is not changed and the organization is enhanced by integrating newly produced strings that are beneficial to the organization. The occurrence of long-lasting stable states interrupted by brief periods of rapid change is a phenomenon known as punctuated equilibrium, and it has been observed in both natural and artificial systems [21].

There are several properties of binary string automata which have been repeatedly observed, including structural similarity between parent and child strings, sequence variation mainly occurring on the edges of strings, passive replication occurring often and active replication being rare [21, 26, 34].

2.3.4 Polymer Chemistry

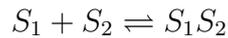
Polymer chemistries have been used to study the emergence and evolution of autocatalytic metabolic networks [19], and self-maintaining metabolisms

 2.3. Formal frameworks for artificial chemistries

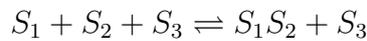
have been observed emerging in polymer chemistries without needing template based reactions [24]. The molecules of the system are character sequences and the reaction rules are composed of concatenation and cleavage reactions.

Objects (S): Polymer chemistries make use of objects that are character sequences made up of molecules from a finite set $S \in \{a, b, \dots\}$, e.g. $M = \{a, b, aa, ab, ba, bb, aaa, aab, \dots\}$.

Reaction rules (R): The reactions that occur in the system are concatenation and cleavage. There are two time-scales on which reactions occur, fast and slow. Random concatenations occur slowly and do not require a catalyst:



Catalysed reactions, molecule cleavage, inflow and decay occur as fast reactions:



Catalysed reactions are not fully dependant on the structure of the reactants, and molecules are assigned randomly as catalysts for reactions.

Algorithm (A): The algorithm simulates a well stirred reaction vessel that is free of any topological structure. The reactor uses a meta-dynamical ODE framework that adapts to changes like new molecules appearing or present species being out-competed [19, 26].

Fig. 2.5 shows the evolution of an autocatalytic polymer network. There is a continuous inflow of molecules a and b . All molecules are subject to a dilution flow, or decay, which is the probability of an entire character sequence being randomly removed. Fig. 2.5 contains examples of closed and open autocatalytic networks. The group of molecules $\{a, b, aa, ba\}$ form

2.3. Formal frameworks for artificial chemistries

a closed network. In a closed network, reactions catalysed by network components only catalyse reactions that are part of the network. In this way closed networks only produce their own components. This type of closed network is known as an organization. The set of molecules $\{a, aa\}$ is not closed because additional molecules, b and ba , can be created by reactions catalysed by members of the set [19].

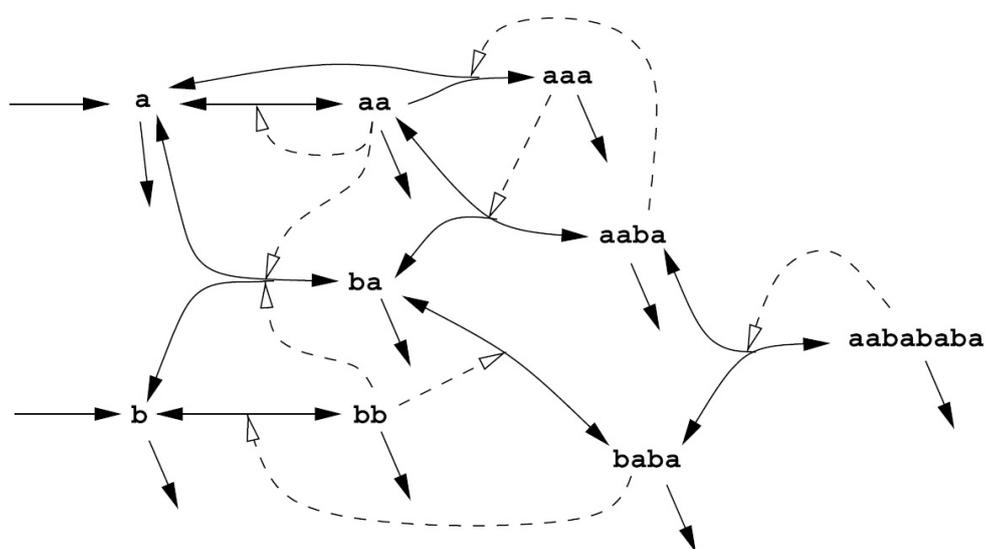


Figure 2.5: The evolution of an autocatalytic polymer network. There is a constant influx of a and b and all reactions are reversible. Catalysed reactions are represented by dotted lines. Downward arrows represent dilution flow.

Constructive polymer chemistries have shown how small, random changes in reaction networks can be amplified by autocatalytic networks or network components. These changes are inherited across generations and can lead to the modification of entire reaction networks [19, 26].

2.3.5 Typogenetics

Typogenetics was first introduced by Hofstadter [31] to illustrate the ‘logic of life’ and the logic underlying the nucleic acid-protein cycle, but his system was incomplete and not computerized [31]. In other attempts [40, 47, 48] typogenetic systems have been completed and computationally adapted in an attempt to implement the ‘molecular logic of the living state’ in artificial systems.

Objects (S): There are two classes of objects: character sequences or ‘strands’, made up from the character alphabet $S \in \{A,C,G,T\}$, and ‘enzymes’, which are sequences of operations interpreted from duplet sections of strand.

$$ACGTAT \longrightarrow cp \cdot ia \cdot ba$$

There are seven types of operations (cut, eliminate, switch, move, copy, introduce and search) which are accessed via a reference table [47]. Strands are created when an enzyme or enzymes operate on their creator strands to produce new generations of strands. A break duplet (AA) can cause multiple enzymes to be created from one strand, and each enzyme will in turn act on the creator strand [26, 47, 48].

$$ACGTATAAGCCATG \longrightarrow cp \cdot ia \cdot ba + ig \cdot dd \cdot yg$$

Reaction rules (R): The reaction rules govern how the typoribosome maps a given strand into an enzyme, and the application of the enzyme to its creator strand. The role of the typoribosome is assumed by the person or computer that translates the strand into enzyme(s) and applies the enzyme action. Starting from left to right a given strand is broken into duplet sections of characters. These duplet characters correspond to operations in

2.3. Formal frameworks for artificial chemistries

a reference table, and the operations are compiled in the same order as the duplets. In this way the strand is converted into a sequence of operations, known as an enzyme. If the break duplex AA is encountered, production of the current strand ends and translation of a new enzyme begins. In this way several enzymes can be created from one strand. Character duplets also have corresponding directional units, or *specificity numbers*, that are all used in determining the binding specificity of the enzyme to either A, C, G or T on the parent strand. Once the enzyme is bound to the strand it begins sequentially executing the operations it is composed of. Any operations that are not possible are not executed, and the strand or strands that remain after the last operation are the next generation of strands [26, 47, 48].

Algorithm (A): The algorithm forms part of the typoribosome, as it is responsible for the translation of the strand into an enzyme. It is also responsible for the application of the enzymes on their respective strands, and the repetition of this process. Strands are measured in generations: the next generation of strands is created when an enzyme operates on its creator and produces a new strand. Multiple strands in a single generation can be occur because of multiple enzymes or double strand sequences, which can be created with the *copy* operation. Systems also usually have a maximum length for strands: because strands have the potential to be infinitely long situations may arise where simulations become to computationally taxing and even impossible due to the physical amount of memory required [31, 39, 47].

Typogenetic systems have been used to investigate the process of self-replication and self-organization from a pre-biotic point of view [14, 47, 48]. Varetto [47] used a completed version of the typogenetic system created by Hofstadter [31] to search for self-replicators by examining the shortest

2.3. Formal frameworks for artificial chemistries

strings first (duplets and triplets). He found that GC and its daughter string GGC were suitable candidates. Fig. 2.6 shows the genealogical tree for the first 43 generations starting from GC as generation 0, in which GC is represented as a solid dot. A new GC first appears in generation 28; this

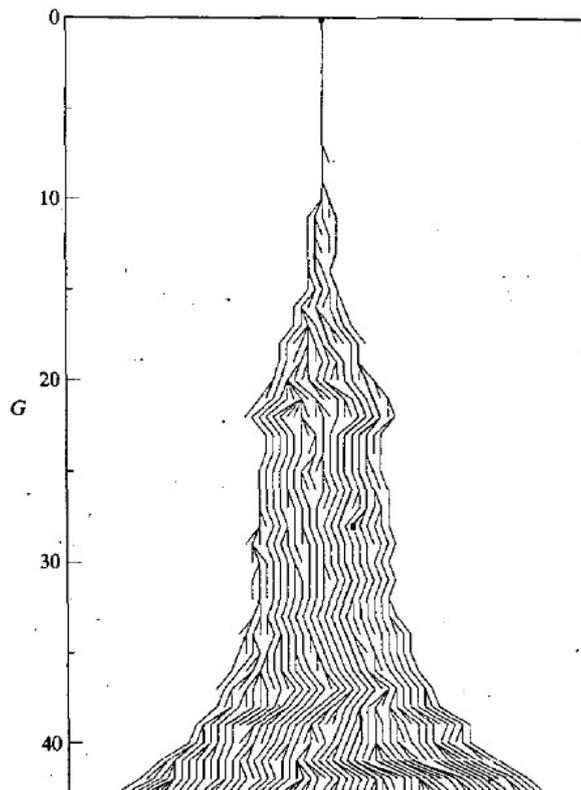


Figure 2.6: Genealogical tree for the first 43 generations of a system initiated with GC (generation 0). GC is produced again on generations 28 and 43. Maximum length 159 [47].

classifies GC a self-perpetuator. In self-perpetuation the parent strand is created more than one generation after the first replication, whereas in self-replication the parent strand is created immediately. GC appeared again in generation 43 and two GC strands appeared in generation 45 (not shown) [47]. More GC strands are produced in subsequent generations and the

2.3. Formal frameworks for artificial chemistries

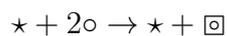
system enters an exponential growth phase. The growth rate corresponds to the maximum length allowed, with systems below maximum length 36 dying out quickly. These systems showed how self-perpetuation can appear as an emergent behaviour of complex systems [47]. In [48] the self-perpetuation of families of strands were shown to be types of self-reproducing autocatalytic metabolisms. These families were even considered as acellular beings, with familial information being conserved in a dynamical loop [48].

2.3.6 Explicit reaction networks: Explicit spatial reaction system

Varela *et al.* [46] defined an explicit spacial network reaction system to explain their concept of autopoiesis. Their concept defines a unity as a network of components that participate in producing and generating components of the production network, to the point where the production network is realized as a distinguishable entity in the given universe [46].

Objects (S): Abstract symbols without defined structure e.g. substrate \circ and catalyst \star elements. These elements occupy space on a 2-dimensional grid with few catalyst \star objects and many \circ molecules. A third element, the \square , can be created by the reaction mechanism [46].

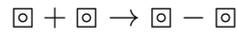
Reaction rules (R): Explicitly defined reaction rules that use structure-less objects. Reactions rules include composition, concatenation and disintegration reactions. In the composition reaction the catalyst \star operates on two nearby \circ elements to produce the \square object.



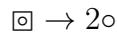
The concatenation reaction states that if one \square is adjacent to another \square

2.3. Formal frameworks for artificial chemistries

they will link together forming a chain [46].



The disintegration rule states that the \square may randomly decay into two \circ objects. This decay can be viewed as spontaneous or as a result of collision with a \square object.



Algorithm (A): In every iteration the \circ and \star objects are allowed to move to one of the surrounding eight adjacent blocks. Two adjacent \circ objects that are next to a \star object become one \square object and an empty space. If two \square objects are next to each other they link together, forming a chain. \circ objects are able to pass through linked $\square - \square$ objects, but \star and \square objects are not able to pass through $\square - \square$ links. New \circ objects are allowed to enter the system to replace empty spaces [46].

Fig. 2.7 shows the first seven instants of a computer simulation. \square molecules form and begin linking together in chains. An empty space is created around the catalyst \star , and new \circ objects that drift in are converted to new \square objects. At instant $t = 4$ the catalyst \star has become completely enclosed by \square objects, but they have not yet fully linked together. At this time there are also \square objects that are enclosed by the $\square - \square$ chain, and by instant seven the \square chain completely closes in on itself, trapping three \square objects inside.

Fig. 2.8 shows the same system at a later stage of the same run. At instant $t = 44$ the functional state is very similar to Fig. 2.7 state $t = 6$, where the catalyst \star is surrounded by an enclosed $\square - \square$ linked membrane. Inside the boundary two \square objects and a \circ object are found. The \circ object

2.3. Formal frameworks for artificial chemistries

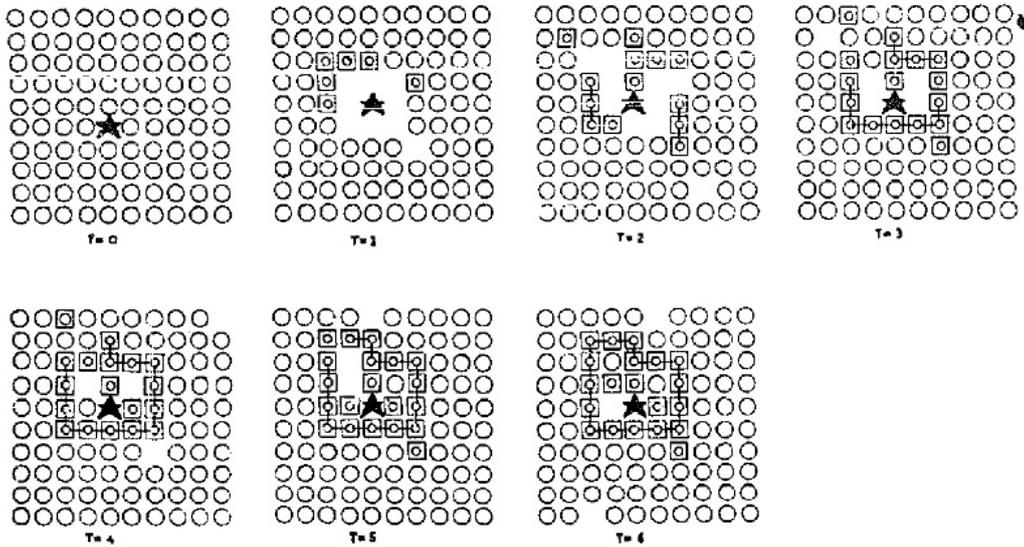


Figure 2.7: The first seven instants, $0 \rightarrow 6$, of one computer run. The interactions between \circ and \star produce chains of $\square - \square$ objects, eventually enclosing the catalyst \circ , and forming a unity closed in terms of efficient causation, but open to material components [46].

can leave through the boundary, but it will be converted into another \square object if it and another \circ are in range of a catalyst \star . At time point $t = 45$ one of the membrane \square objects randomly disintegrates creating an opening in the \square membrane. Internal \square objects move randomly inside the membrane, while \circ objects entering through the breach are catalysed to \square objects. When \square objects move next to open membrane ends they are concatenated to the existing membrane, and by instant $t = 47$ the membrane had restored itself to a closed state [46].

2.3.7 Lambda Calculus

Lambda (λ)-calculus has been used to define abstract constructive chemistries that have implicit structure-function mappings. The λ -calculus framework allows normalized λ -expressions to be interpreted as operators acting on

2.3. Formal frameworks for artificial chemistries

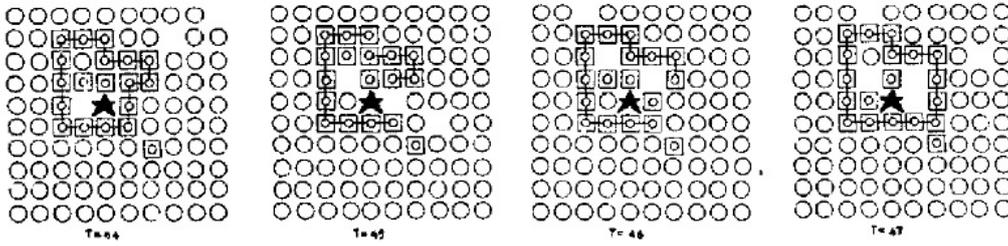


Figure 2.8: Instants $44 \rightarrow 47$, of one the same computer run. The catalyst \star is completely enclosed by $\square - \square$ objects. When the membrane ruptures it is quickly restored by internal \square molecules, demonstrating the self-fabrication of the unity [46].

other λ -expressions.

Objects (S): in λ -calculus objects are normalized λ -expressions. A λ -expression is a word over an alphabet $A = \{\lambda, ., (,)\} \cup V$, where $V = \{x_1, x_2, \dots\}$ is an infinite set of variable names. The set of λ -expressions S is defined for $x \in v$, $s_1 \in S$, $s_2 \in S$ by

$x \in S$: variable name

$\lambda x.s_2 \in S$: abstraction

$(s_2)s_1 \in S$: application

The abstraction, $\lambda x.s_2$ can be interpreted as a function definition, where x is the parameter in s_2 and where s_2 is the body of the function. The expression $(s_2)s_1$ can be interpreted as the application of s_2 on s_1 [19, 26].

Reaction rules (R): This application of s_2 on s_1 is the application of a function s_2 on data s_1 , and it is formalized by the rewriting rule, also known as the β -rule:

$$(\lambda x.s_2)s_1 = s_2[x \leftarrow s_1]$$

where $s_2[x \leftarrow s_1]$ is part of the reaction mechanism responsible for replacing every unbound occurrence of x in s_2 by s_1 . A bound occurrence of x

 2.3. Formal frameworks for artificial chemistries

appears appears in the form $\dots (\lambda x \dots x \dots) \dots$. If the variable x becomes bound the β -rule may no longer be applied. An example of interaction, let $s_1 = \lambda x_1.(x_1)\lambda x_2.x_2$ and $s_2 = \lambda x_3.x_3$. From this we can derive:

$$(s_1)s_2 \implies (\lambda x_1.(x_1)\lambda x_2.x_2)\lambda x_3.x_3 \implies (\lambda x_3.x_3)\lambda x_2.x_2 \implies \lambda x_2.x_2$$

The most simple reaction mechanism that can be defined for two colliding objects, s_1 and s_2 , is for s_1 to operate on s_2 . This is done with the *normalForm* procedure, which reduces the argument term to normal form.

$$s_1 + s_2 \longrightarrow s_1 + s_2 + \mathit{normalForm}((s_1)s_2)$$

The λ -calculus also allows for a generalization of the collision rule by the λ -expression $\Phi \in S$:

$$s_1 + s_2 \longrightarrow s_1 + s_2 + \mathit{normalForm}(((\Phi)s_1)s_2)$$

This process is bound by time and memory, and the reaction may not be complete before the memory is exceeded, which will result in an unstable term and no reaction products [19, 24, 26].

Algorithm (A): The algorithm used by [28, 29] simulates a well stirred population undergoing second-order mass-action kinetics. In one iteration two random members of the population are selected. One is applied to the other according to the reaction mechanism, and the product replaces a random member of the population. Populations typically range between 1000-3000 members.

It was noted by Fontana and Buss [29] that the diversity of the initial population often rapidly reduced to only one or two self-replicating species.

 2.3. Formal frameworks for artificial chemistries

This type of surviving structural organization was termed a level-0 organization, and was defined by a few closely coupled replicators. To encourage more complex networks, filter conditions were introduced that would nullify reactions that produce one of the reactants, meaning no self replication was allowed. As a result of this limitation the term level-1 organization was created to define the emergent organization.

Level-1 organizations consist of a very large and potentially infinite number of members, however, because the reactor is finite only a portion of the organization is usually represented. As a result the level-1 organization constantly cycles through its members, creating new species and erasing old members, and then repeating the loop. It was found that newly inserted species would rarely be incorporated into the organization, but when interaction occurred the organization branched into different layers.

Level-2 organizations were described as two coexisting level-1 organizations. The coexistence was characterised by two factors: the interaction of the organizations through an intermediate species, and the moderation of the interaction by the intermediate species. Level-2 organizations rarely evolve spontaneously from random populations, but they can be synthetically generated through the merging of two independently evolved level-1 organizations [19, 24, 26, 28, 29]. Fontana [28] also found that members of organizations that survived the transient phase have similar syntactical structure. All of the expressions in Table 2.1 share at least two functions:

Table 2.1: Examples of λ -expressions with syntactic similarity.

λ -chemistry
$\lambda x_1. \lambda x_2. \lambda x_2$
$\lambda x_1. \lambda x_2. \lambda x_3. \lambda x_2$
$\lambda x_1. \lambda x_2. \lambda x_3. \lambda x_3$
$\lambda x_1. \lambda x_2. \lambda x_3. \lambda x_4. \lambda x_3$

2.4 Application of artificial chemistries

ACs have proved to be powerful tools when researching artificial life (AL). Using integrated interdisciplinary methods, AL research attempts to extract the first principles of life by using systems which are modelled from examples of living organisations [24, 26, 37, 41, 42]. ACs are also useful when studying the possible origins of metabolic networks and related problems, such as how RNA, DNA and protein metabolisms came into existence from pre-biotic circumstances [8, 36]. The idea that RNA played a pivotal role in the origin of life is of particular importance. It is known that RNA, like DNA, can act as information carrier and can fold into a 3-dimensional form that can act auto-catalytically on itself [3]. The produced strands may be the same as the operator, i.e. the strand operated on, or may be a new sequence of RNA. The AC of binary strings mentioned above is particularly useful in simulating such RNA-based networks [6]. ACs thus allow us to theoretically investigate chemical and pre-biotic evolutionary scenarios, such as the formation of complex organic molecules from simple inorganic compounds. The study of pre-biotic scenarios thus relies on physical experimentation and theoretical simulation, because there is no pre-biotic fossil record [19, 36]. The ability of ACs to replicate the constructive behaviour of chemical systems is the main advantage of using ACs when reproducing chemical evolution in artificial systems.

Chapter 3

Binary String Systems

3.1 Binary Sequences

Just as the information on how to make proteins is encoded in DNA in the form of a sequence of N-bases, the information stored in all of our electronic devices is encoded in binary format [8, 33]. All electronic information we have can be broken down into ones and zeros, known as bits [4]. The bits are ‘stored’ in hardware components known as transistors, which act as electronic switches [33]. If current flows through the transistor it is on and holds a value of ‘one’. If current does not flow through the transistor it is off and holds a value of ‘zero’ [33]. Binary sequences thus have physical locations in hardware components. Programs, which are software, are made up of specific binary sequences [8]. Programs, in general, read data from a source, process the data internally and then output the modified data. On the hardware level this translates into a very specific arrangement of bits, B_P , acting (operating) on an initial data set, D_1 , and producing a second data set, D_2 [8]:

$$B_P \oplus D_1 = D_2$$

where \oplus is the operator symbol.

This formula may remind us of a biochemical reaction in which an enzyme or catalyst acts on a substrate to produce a product without itself being consumed or modified in the process. This would be the case if D_1 were overwritten by the operator B_P and then replaced with D_2 . However, unlike the real situation where the conversion of substrate D_1 to product D_2 can occur spontaneously over time, albeit slowly, this will not happen in the digital case [6]. If D_1 were to be first duplicated and then modified, B_P would be more similar to a copier than to an enzyme, although it would be a true copier only if it did not modify D_1 to D_2 .

So how precisely can bit-sequences and programs be related to enzyme catalysts, copiers and self-replicators? Binary sequences, like RNA, act as carriers of information, and sequences can be folded into the 2-dimensional form of a matrix [1, 6, 8]. A matrix can interact ‘catalytically’ with itself or can act on other binary sequences. Binary sequences normally occur in the 1-dimensional ‘string’ form and require a mapping in order to fold into a 2-dimensional ‘catalytically active’ form [1, 4, 6]. Most binary sequences can be folded into a 2-dimensional matrix. If the length of a binary sequence can be divided by something other than itself or one, it can be folded into a 2-dimensional matrix [2]. Binary strings that are prime numbers therefore cannot be folded into a 2-dimensional form. A string can only fold into a square matrix if the square root of the sequence length (N) is a whole number [1, 2]:

$$N \in [1, 4, 9, 16, \dots]$$

Binary strings can be written in their shorthand form where they are named after their corresponding base₁₀ number, i.e., s_{11} represents string [1011] (base₂) or 11 (base₁₀). A string that has been folded into a matrix

3.1. Binary Sequences

can act as an operator. The operator symbol \oplus is used to denote the act of such a folded string operating on another [1, 2]. In the following example folded string s_x operates on string s_y to produce string s_z :

$$s_x \oplus s_y = s_z$$

The AC we chose to examine first was the 4-bit binary string system. We chose this system because 4-bit binary strings are the shortest binary strings that can be folded into square matrices [2–4]. The single bit string system is the only binary string system shorter than 4-bit, but it is a trivial system as it does not yield complex behaviour [2, 3]. This is because there are only two string species, $s_0=[0]$ and $s_1=[1]$, yielding only 4 possible operations in this system:

$$s_0 \oplus s_0 = s_0$$

$$s_1 \oplus s_0 = s_0$$

$$s_0 \oplus s_1 = s_0$$

$$s_1 \oplus s_1 = s_1$$

While the above 1-bit system only has two species the 4-bit system has 16 different species ranging from s_0 to s_{15} [1, 2]:

$$\begin{array}{cccc} s_0=[0000] & s_1=[0001] & s_2=[0010] & s_3=[0011] \\ s_4=[0100] & s_5=[0101] & s_6=[0110] & s_7=[0111] \\ s_8=[1000] & s_9=[1001] & s_{10}=[1010] & s_{11}=[1011] \\ s_{12}=[1100] & s_{13}=[1101] & s_{14}=[1110] & s_{15}=[1111] \end{array}$$

3.2 Sequence Folding: Mapping

The 1-bit system is also simple because there is no mapping required from string to operator, string S_1 is [1] and operator S_1 is also [1]. With strings longer than one bit a mapping is required to go between string and operator [1, 2]. Consider the string $s_x = [4321]$. Note that s_x does not represent the value of four thousand three hundred and twenty one; 1, 2, 3, and 4 represent positions in the string. The reason for the right-to-left ordering of the numbers is that binary number strings fill up from the right. In a column vector (4×1) representation of such a string position 1 is at the top, as follows:

$$s_x = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

A mapping F for string $s_x = [4321]$ into a 2x2 matrix can occur in the following way:

$$F : s_x \rightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

This is an example of *standard canonical folding*. A mapping always starts in the top left corner [1, 2, 4]. With canonical folding the string s_x is broken up and read into the matrix from top-left to top-right, top-right to lower-left, then from lower-left to lower-right:

$$F_{can} : s_x = \begin{bmatrix} 1 & \rightarrow & 2 \\ & \swarrow & \\ 3 & \rightarrow & 4 \end{bmatrix}$$

In the 4-bit binary system there are only four types of mapping that we examined, including canonical folding. The other mappings examined were transposed canonical, topological and transposed topological folding. In

3.2. Sequence Folding: Mapping

linear algebra a matrix is transposed when it is reflected over its main diagonal, which runs from the top-left to bottom-right. Aptly named transposed canonical folding is the transposition of canonical folding. Compared with canonical folding the positions of ‘2’ and ‘3’ have been swapped around. This causes the matrix to proceed from top to bottom then left to right [1, 2, 4].

$$F_{t-can} : s_x = \begin{bmatrix} 1 & & 3 \\ \downarrow & \nearrow & \downarrow \\ 2 & & 4 \end{bmatrix}$$

The next two matrices are somewhat different from the previous two in that they fold topologically and not canonically. Topological folding is more representative of biological folding, starting at the top left and then proceeding left to right, top to bottom and then right to left.

$$F_{top} : s_x = \begin{bmatrix} 1 & \rightarrow & 2 \\ & & \downarrow \\ 4 & \leftarrow & 3 \end{bmatrix}$$

The last mapping is the transpose-topological type. The method of transposition is the same as it was with canonical folding: the matrix is reflected over its main diagonal. The matrix is built from top to bottom, left to right and then bottom to top.

$$F_{t-top} : s_x = \begin{bmatrix} 1 & & 4 \\ \downarrow & & \uparrow \\ 2 & \rightarrow & 3 \end{bmatrix}$$

These four mappings were chosen because they represent possible biological folding types. If you were to imagine an RNA sequence, say $s_{RNA} = [UGCA]$, that sequence would be held together by a backbone, a ribose-5-phosphate backbone in the case of RNA: $s_{RNA} = [U \leftarrow G \leftarrow C \leftarrow A]$.

3.2. Sequence Folding: Mapping

We wanted to allow the binary sequences to only fold in ways that RNA sequences could fold into, thus no bonds may cross over each other as this would represent bonds going through each other. For those reasons the following mappings were not examined:

$$F_{crossfold_1} : s_{RNA} = \begin{bmatrix} A & & G \\ & \times & \uparrow \\ U & & C \end{bmatrix}$$

$$F_{crossfold_2} : s_{RNA} = \begin{bmatrix} A & & U \\ & \times & \\ G & \leftarrow & C \end{bmatrix}$$

Although there were only four mappings chosen, in Chapter 4 we examine systems that use six different folding methods. The cross-folding methods are not examined. Two combinatorial methods are used alongside the four ‘normal’ folding types, ‘random folding’ and ‘selective folding’.

Systems that utilize random folding map operators into one of the four ‘normal’ types. Every iteration a folding type is randomly selected from one of the four types and used for the operator mapping. The folding types are not used sequentially, and some folding types will be selected slightly more than others. The longer a simulation, the more equal the folding type selection will be.

Systems that utilize selective folding map operators into one of the four ‘normal’ types based on ‘string memory’. With every iteration, a string is selected to be an operator. To choose a folding type, the ‘string memory’ of the selected string is consulted. For each folding type, the ‘string memory’ records the percentage likelihood that the string has of folding into that particular type. All folding types are assigned an initial weight with the value of 2.5% of the total population; this determines how fast or slow it gains preference for folding types. 2.5% was arbitrarily chosen after testing

3.3. Binary String Interactions

several values, ranging from 0% to 100% of the population. It was found that at 2.5%, for any string that actively replicates with only one folding type, that folding type could increase from 25% selection chance to 33% selection chance over the first generation. This rate of change was deemed gradual enough, with strings being able to adapt to change quick enough to survive, but not that rapidly so as to quash other folding types. Each string has one memory bank for each folding type. Every iteration, if the reaction is an active replication (operator producing itself), the memory bank for the folding type of the operator will grow by one point. The more a folding type benefits a string, the more its memory bank for that folding type will grow, increasing the chances of it being selected every iteration.

3.3 Binary String Interactions

Binary strings interact with each other in the system through an imposed algorithm. Two sequences are selected randomly from a population of strings. One of the strings is then folded via a mapping into an operator. The operator then acts on the unfolded string. The method of action of the operator is standard matrix multiplication [1, 2, 4]. In the following example $s_a = [DCBA]$ and $s_x = [4321]$ interact with each other, with s_a as operator:

$$s_a \oplus s_x = s_{new}$$

$$F_{can} : s_a = \begin{bmatrix} A & \rightarrow & B \\ & \swarrow & \\ C & \rightarrow & D \end{bmatrix}$$

$$s_a \oplus s_x = \begin{bmatrix} A & B \\ C & D \\ A & B \\ C & D \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} A * 1 + B * 2 \\ C * 1 + D * 2 \\ A * 3 + B * 4 \\ C * 3 + D * 4 \end{bmatrix} \rightarrow \begin{bmatrix} A * 1 + B * 2 \\ C * 1 + D * 2 \\ A * 3 + B * 4 \\ C * 3 + D * 4 \end{bmatrix} = s_{new}$$

3.3. Binary String Interactions

$$s_{new} = [C * 3 + D * 4, A * 3 + B * 4, C * 1 + D * 2, A * 1 + B * 2]$$

The above example may seem intimidating, however, if we replace s_a and s_x with binary sequences, $s_8 = [1000]$ and $s_7 = [0111]$, the binary process is more understandable:

$$s_8 \oplus s_7 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 * 1 + 0 * 1 \\ 0 * 1 + 1 * 1 \\ 0 * 1 + 0 * 0 \\ 0 * 1 + 1 * 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = [0010] = s_2$$

The equation above illustrates how the operator acts on regions of two bits at a time. It does not matter if the operator starts from one side or the other, as long as the new string is written into the register in the same order as the original string was read out. In the following example s_3 operates on s_7 and produces s_5 :

$$s_3 \oplus s_7 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 * 1 + 1 * 1 \\ 0 * 1 + 0 * 1 \\ 1 * 1 + 1 * 0 \\ 0 * 1 + 0 * 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 1 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = [0101] = s_5$$

In the above example, in the step after multiplying out, the string [0102] appears. In the following step it is converted to [0101]. This conversion is necessary to keep the system in a binary format. It is possible to have systems that start binary and allow for such reactions to occur, but then they are no longer binary systems, and we are not interested in examining them here. In the reaction $s_8 \oplus s_7 = s_2$ and in the reaction $s_{10} \oplus s_7 = s_{10}$ new strings were produced. These reactions include one of the five classes

of reaction types:

$$s_x \oplus s_y = s_z(1)$$

$$s_x \oplus s_x = s_y(2)$$

$$s_x \oplus s_x = s_x(3)$$

$$s_x \oplus s_y = s_x(4)$$

$$s_x \oplus s_y = s_y(5)$$

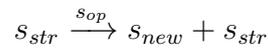
The first reaction (1) is a new string producing reaction where two dissimilar strings interact and produce a new string species. The second reaction (2) is also a new string producing reaction where two identical strings interact and produce a new string. The third reaction (3) is an example of self-replication where two identical strings interact to produce a third identical string. The fourth (4) reaction type, where two dissimilar strings interact to produce the operator, is known as active replication. The fifth (5) reaction type, where two dissimilar strings interact to produce the string, is known as passive replication. These five reaction classes can be further grouped into three more general reaction types: new string generation (1 + 2), self-replication (3), and replication (4 + 5) reactions [3, 4].

3.4 Model design

In the previous section the method of binary string interaction was described. However the description was only of a single interaction between two strings. To understand string interaction and metabolism on a large scale we need more strings and a mechanism to drive interactions. This section briefly describes the original model, designed by Banzhaf [4], and compares it with our model of the same system. Both systems have similar base requirements:

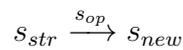
3.4.1 Population dynamics

Binary strings need a ‘space’ to exist in. Binary systems hold a finite number of strings, known as the population or soup with size (M). The population does not need to stay constant throughout the simulation. A system is initialized with M strings and newly produced strings may replace existing strings or increase the population. Thus a population is able to grow from an initial state M_1 to a final state M_2 . If the population is to be kept constant a string needs to be removed for every string added. There are three general methods for removing strings, each representing a different type of simulation: (I) Deletion of a random string from the population. This method removes strings according to their frequency in the population. This method can be seen as a reaction vessel with an overflow mechanism, with the addition of the new string ($s_{op} \oplus s_{str} = s_{new}$) replacing a random string (s_{ran}).



$$M = M + s_{new} - s_{ran}$$

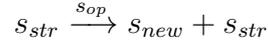
(II) Removal of the string that was acted on. This can be chemically equated to the substrate (string) being acted on by the enzyme (operator) and being transformed into the product (new string).



$$M = M + s_{new} - s_{str}$$

(III) The third option is to remove the operator from the soup. Here the operator can no longer be classified as a catalyst as it is consumed in the reaction by acting on the ‘substrate’ string and converting itself into the

new product string.



$$M = M + s_{new} - s_{op}$$

3.4.2 Lethal strings

A second important factor for both systems is monitoring the population for potentially lethal strings, including the *destructor* and the *exploiter*. Lethal strings are able to replicate with almost all strings in any operator conformation, as a consequence they quickly dominate the population and extinguish other string types. Destructor strings consist only of zeros (s_0) and always produce themselves regardless of being the operator or string. To counter their destructive nature they are not generated with the initial string population and they are not added to the soup if they are produced by a reaction. Thus when a reaction between two strings produces a destructor the reaction is considered elastic and does not occur. Exploiter strings consist of only ones (s_1) and are able to self-replicate and replicate with a large number of strings and often dominate the population. A substitution probability (P_k) can be introduced to counter their exploitive nature. The substitution algorithm examines a random string (k) and compares the number of 1 elements (I_k) with the string length (N) and substitutes according to the probability percentage:

$$P_k = [I_k/N]^n * 100$$

The parameter n serves as an adjustable threshold for string decay. The exploiter, consisting only of 1's will always have $I_k = N$ and $P_k = 100\%$, regardless of n . Substituted strings are replaced with the copy of a string

randomly selected from the soup. In essence this makes S_{15} highly mutagenic, meaning it has a chance to decay into a random soup member. The substitution probability also has a chance of replacing strings other than the exploiter. If $n = 1$ then there is 75% chance of strings with three 1's being substituted ($P_k = [3/4]^1 * 100$). This could be problematic because the probability is only meant to counter the exploiter string. One way to counter this property is to set n to a very high value as this lowers the probability of a string other than the exploiter being substituted:

n	$P_{k:I_k=3}$	$P_{k:I_k=2}$	$P_{k:I_k=1}$
10^0	75%	50%	25%
10^1	5.6%	0.1%	$10^{-4}\%$
10^2	$\approx 0\%$	$\approx 0\%$	$\approx 0\%$

A more severe way of dealing with the exploiter string is to treat it the same as the destructor by constantly monitoring the soup and removing it upon appearance. This does however impose more limitations on a system that is supposed to be open to its own modifications as much as possible.

3.4.3 Model comparison

Binary systems require an algorithm to drive string interactions. The algorithm used by Banzhaf et al. consists of several steps in specific sequence:

1. A population of M strings is generated.
2. A random string is selected and folded into an operator.
3. The operator is applied to a second randomly selected string, generating a third string.
4. The operator is unfolded and all three strings are released back into the soup, unless the produced string is a destructor.

5. Remove a random string from the soup in order to compensate for the addition in step 4
6. Select a random string and substitute it according to the probability constant.
7. Proceed to step 2

One interaction or iteration occurs from steps 2–4, with step 7 repeating the process. One generation has passed if a vessel goes through M iterations and has a population size of M . Time in binary systems is often measured in generations, with smaller systems (population 10^3 – 10^4) often reaching an attractor state between 10 and 100 generations, while larger systems (population $\gg 10^5$) often require hundreds of thousands of generations before reaching a final attractor state.

From step 4 onwards, the binary system we designed differs from the model proposed by Banzhaf [4]:

1. A population of M strings is generated.
2. A random string is selected and folded into an operator.
3. The operator is applied to a second randomly selected string, generating a third string.
4. If the generated string complies with the selection parameters it is added to the population.
5. If the string does not comply then the iteration is re-run.
6. A random string is removed from the population to compensate for the addition in step 4.

7. If substitution is active a random string is evaluated for substitution.
8. Return to step 2.

The main difference in the design of the model algorithms occurs because of a) the different programming languages used and b) the different end requirements (and adaptability) of the model that we required. The changes made to the model allow for additional selection parameters to be applied during string generation in step 4. A minor difference between the models occurs is in the way strings are selected. In the model described by Banzhaf [1], strings are removed from the population when selected for interaction, while in our model the strings selected for interaction are technically never removed from the soup, so they cannot be added back to the soup after interacting. This is different because it is simply unnecessary to remove the strings from the population in our model. The last three steps in each model design are the same: remove a string to compensate for the addition, select a random string and apply the substitution probability constant, and repeat the process by starting again from an earlier step. The last step in each model drives the system forward and essentially creates ‘system time’, allowing the populations to be dynamic and able to change states.

Chapter 4

Results

4.1 Reaction tables

In a 4-bit string system there are 16 different string species. There are 256 different possible reactions if every string operates once on itself and once on every other string. We produced reaction tables (4.3, 4.4, 4.5, 4.6) for each of the four mappings we examined, in the reaction tables are the results of each of the 256 different reactions. The y-axis shows the chosen operator and the x-axis shows the chosen string. We used colour coding on the reaction tables to group reactions of the same type together. Only the four biologically consistent folding types were used to produce the reaction tables. Table 4.1 shows the number of types of reactions for each type of folding examined.

	Self-replication	Replication	Generation
Canonical	5	106	145
Trans-canonical	11	132	113
Topological	5	106	145
Trans-topological	7	110	139

Table 4.1: Reaction types per folding type.

Each reaction table has its own unique pattern of string formation, although there are certain similarities between the reaction tables. The transpose-canonical (4.4) folding type has 11 self-replication reactions, the most of the four types examined, and the canonical (4.3) and topological (4.5) folding types shared the same amount of reaction types. Self-replication reactions always occur on the major diagonal where the operator and string are the same species. It was expected that there would be a different frequency of string species produced for each of the folding types, however, all of the four reaction tables (F_{can} , F_{t-can} , F_{top} and F_{t-top}) contained the same frequency of string species, although they were mostly distributed differently across the reaction tables. Table 4.2 shows string production in frequency and percent: Frequency represents the number of strings produced per reaction table; percent represents the percent each species contributes to the reaction table. Strings were also grouped into categories based on how many times they were produced in each reaction table. Arranging table 4.2 into a 4x4 matrix resulted in a symmetrical matrix. The symmetrical matrix A is equal to its transposed matrix A^T , so that $A = A^T$.

In Table 4.2 the destructor string (S_0) is produced more than any other string type, being produced 49 times in each reaction table. Even though the destructor string is entirely self replicating, all 31 reactions that involve S_0 as either a string or an operator in all four folding types result in the production of S_0 , it is also produced in other 18 reactions that do not involve S_0 as either operator or string. The exploiter string (S_{15}) is also produced at much higher frequencies than other strings, accounting for 35 reactions in each reaction table. It was expected that the more a string species is produced in a reaction table the more likely it will be to survive in different

conditions. It would also be logical to expect that groups of strings with production frequencies higher than other groups would be more likely to survive under different conditions, and would be found at concentrations higher than groups with lower production frequency.

Table 4.2: String production frequency for F_{can} , F_{t-can} , F_{top} and F_{t-top} .

Occurrence	Percent	Group	String
49	19.14 %	Des	S_0
35	13.67 %	Exp	S_{15}
21	8.20 %	1	S_3, S_5, S_{10}, S_{12}
15	5.86 %	2	S_1, S_2, S_4, S_8
6	2.34 %	3	$S_7, S_{11}, S_{13}, S_{14}$
2	0.78 %	4	S_6, S_9

Table 4.2 revealed that the group of strings with the highest production frequency, set 1, contained strings S_3 , S_5 , S_{10} and S_{12} at 8.20% each or 32.80% of the total reaction table. Subsets are not composed of random species and members of subsets are mathematically related objects. Members of this subset all have an equal number of ones and zeros that are asymmetrically distributed across the string: $S_3 = [0011]$, $S_5 = [0101]$, $S_{10} = [1010]$, $S_{12} = [1100]$. The combination of asymmetrical distribution and an equal 1 : 0 ratio creates operators that are horizontally and vertically symmetrical (except for diagonally symmetrical operators F_{top} , F_{t-top} : S_5 , S_{10}). Vertically symmetrical operators, depending on the string they operate on, are likely to produce two-bit strings S_5 and S_{10} and single-bit strings S_1 , S_2 , S_4 , S_8 . Operators with ones occupying the lower hemisphere are capable of producing S_{10} , S_8 and S_2 . Operators with one-bits in the upper hemisphere are capable of producing S_5 , S_4 and S_1 .

$$\begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \rightarrow \begin{bmatrix} 0 \cdot A + 0 \cdot B \\ 1 \cdot A + 1 \cdot B \\ 0 \cdot C + 0 \cdot D \\ 1 \cdot C + 1 \cdot D \end{bmatrix} \quad (4.1)$$

$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \rightarrow \begin{bmatrix} 1 \cdot A + 1 \cdot B \\ 0 \cdot A + 0 \cdot B \\ 1 \cdot C + 1 \cdot D \\ 0 \cdot C + 0 \cdot D \end{bmatrix} \quad (4.2)$$

Horizontally symmetrical operators will produce strings dependant on the A and C or B and D sections of the string, however, both AC and BD string combinations lead to the production of S_3 , S_{12} or S_{15} . Operators with right aligned ones produce these strings by operating on S_2 , S_8 and S_{10} respectively. Left aligned operators produce these strings by operating on S_1 , S_4 and S_5 respectively.

$$\begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \rightarrow \begin{bmatrix} 0 \cdot A + 1 \cdot B \\ 0 \cdot A + 1 \cdot B \\ 0 \cdot C + 1 \cdot D \\ 0 \cdot C + 1 \cdot D \end{bmatrix} \quad (4.3)$$

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \rightarrow \begin{bmatrix} 1 \cdot A + 0 \cdot B \\ 1 \cdot A + 0 \cdot B \\ 1 \cdot C + 0 \cdot D \\ 1 \cdot C + 0 \cdot D \end{bmatrix} \quad (4.4)$$

In table 4.2 the subset with the second highest string production, set 2, contains S_1 , S_2 , S_4 and S_8 at 5.86% each and combined account for 23.44% of the reaction tables. Members of this subset all contain a single one value in their sequence: $S_1 = [0001]$, $S_2 = [0010]$, $S_4 = [0100]$, $S_8 = [1000]$. These strings will form operators that contain a single one bit in one of the four quadrants, dependant on the string and folding type of the operator. Operators of this type often produce themselves or another single bit string,

and at the most they can produce two bit strings $S_5 = [0101]$ and $S_{10} = [1010]$.

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \rightarrow \begin{bmatrix} 0 \cdot A + 0 \cdot B \\ 0 \cdot A + 1 \cdot B \\ 0 \cdot C + 0 \cdot D \\ 0 \cdot C + 1 \cdot D \end{bmatrix} \quad (4.5)$$

$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \rightarrow \begin{bmatrix} 0 \cdot A + 0 \cdot B \\ 1 \cdot A + 0 \cdot B \\ 0 \cdot C + 0 \cdot D \\ 1 \cdot C + 0 \cdot D \end{bmatrix} \quad (4.6)$$

$$\begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \rightarrow \begin{bmatrix} 0 \cdot A + 1 \cdot B \\ 0 \cdot A + 0 \cdot B \\ 0 \cdot C + 1 \cdot D \\ 0 \cdot C + 0 \cdot D \end{bmatrix} \quad (4.7)$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \rightarrow \begin{bmatrix} 1 \cdot A + 0 \cdot B \\ 0 \cdot A + 0 \cdot B \\ 1 \cdot C + 0 \cdot D \\ 0 \cdot C + 0 \cdot D \end{bmatrix} \quad (4.8)$$

In table 4.2 the group of strings with the second lowest production frequency, set 3, contains S_7 , S_{11} , S_{13} and S_{14} at 2.34% each or 9.36% total of the reaction tables. These strings all contain three one bits: $S_7 = [0111]$, $S_{11} = [1011]$, $S_{13} = [1101]$ and $S_{14} = [1110]$. These strings form operators that only contain a single zero bit in one of their quadrants. This group of operators produce a large number of different string types, and they have the lowest potential for self-replication.

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \rightarrow \begin{bmatrix} 0 \cdot A + 1 \cdot B \\ 1 \cdot A + 1 \cdot B \\ 0 \cdot C + 1 \cdot D \\ 1 \cdot C + 1 \cdot D \end{bmatrix} \quad (4.9)$$

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \rightarrow \begin{bmatrix} 1 \cdot A + 0 \cdot B \\ 1 \cdot A + 1 \cdot B \\ 1 \cdot C + 0 \cdot D \\ 1 \cdot C + 1 \cdot D \end{bmatrix} \quad (4.10)$$

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \rightarrow \begin{bmatrix} 1 \cdot A + 1 \cdot B \\ 0 \cdot A + 1 \cdot B \\ 1 \cdot C + 1 \cdot D \\ 0 \cdot C + 1 \cdot D \end{bmatrix} \quad (4.11)$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \rightarrow \begin{bmatrix} 1 \cdot A + 1 \cdot B \\ 1 \cdot A + 0 \cdot B \\ 1 \cdot C + 1 \cdot D \\ 1 \cdot C + 0 \cdot D \end{bmatrix} \quad (4.12)$$

In table 4.2 the group of strings with the lowest production frequency, set 4, contains only two members, S_6 and S_9 , at 0.78% each or 1.56% total of the reaction tables. These strings contain two one bits that can form horizontally, vertically and diagonally symmetrical operators: $S_6 = [0110]$, $S_9 = [1001]$. Together these two operators produce the largest number of different string types, and when placed together they have a very high potential for self-replication.

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \rightarrow \begin{bmatrix} 0 \cdot A + 1 \cdot B \\ 1 \cdot A + 0 \cdot B \\ 0 \cdot C + 1 \cdot D \\ 1 \cdot C + 0 \cdot D \end{bmatrix} \quad (4.13)$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \rightarrow \begin{bmatrix} 1 \cdot A + 0 \cdot B \\ 0 \cdot A + 1 \cdot B \\ 1 \cdot C + 0 \cdot D \\ 0 \cdot C + 1 \cdot D \end{bmatrix} \quad (4.14)$$

The four reaction tables and table 4.2 were used to find initial systems to examine. It was decided that two groups of systems, homogeneous and heterogeneous, would be examined. Homogeneous systems are initiated with only one string type. Heterogeneous populations are initiated with

multiple string types. Although heterogeneous systems may be initiated with any subset of initial strings, we only examined heterogeneous systems initiated with all string types.

The first group of reactions we examined were simple homogeneous systems. These systems were initiated with one string type and strings could only fold in one folding type throughout the entire reaction.

Random (F_{ran}) and selective (F_{sel}) homogeneous systems were examined second. They were initiated with one string type and these systems allow either random or selective folding for the duration of the reaction. Random folding selects a random folding type for the chosen string. Selective folding uses a mechanism to rank folding types and promote folding that is beneficial to the selected string. Selective folding initially weights all folding types equally, when an operator replicates itself the current fold conformation is promoted. This increase raises the chance of the promoted folding type being selected again. Promotion is string specific: if S_6 folds F_{t-top} and operates on S_9 to produce S_6 , then F_{t-top} is promoted for S_6 and the next time S_6 folds into an operator F_{t-top} is more likely to be selected.

Heterogeneous systems were examined third. These systems were initiated with all string types, except the destructor string. Four sets of experiments were created, and each set used all six folding types (F_{can} , F_{t-can} , F_{top} , F_{t-top} , F_{ran} and F_{sel}). The four sets were differentiated by the manner in which the exploiter string (S_{15}) was dealt with. In set 1 the S_{15} was not regulated and was allowed to replicate freely with any string. In set 2 S_{15} was initiated with the population, but reactions that produce the exploiter string were considered elastic and did not occur. As a direct result S_{15} was quickly consumed by other members of the population. In set 3 the probability constant, P_k , was introduced, with the threshold n set at 10^2 .

4.1. Reaction tables

This meant that every iteration a random member of the population would be examined, if the examined string was S_{15} then it would be replaced with a random member of the population, essentially a mechanism lighter than making the reactions completely elastic. In set 4 the probability constant threshold n was set to 10^0 . This meant that even if the examined string was not S_{15} it still had 25% chance to decay for every one bit it contained, giving every string a chance to decay upon random selection.

Table 4.3: Canonical folding, 5 self-replications, 106 replications, 145 new string reactions.

15	0	3	3	3	12	15	15	15	12	15	15	15	12	15	15	15
14	0	2	3	3	8	10	11	11	12	14	15	15	12	14	15	15
13	0	3	2	3	12	15	14	15	8	11	10	11	12	15	14	15
12	0	2	2	2	8	10	10	10	8	10	10	10	8	10	10	10
11	0	1	3	3	4	5	7	7	12	13	15	15	12	13	15	15
10	0	0	3	3	0	0	3	3	12	12	15	15	12	12	15	15
9	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	0	0	2	2	0	0	2	2	8	8	10	10	8	8	10	10
7	0	3	1	3	12	15	13	15	4	7	5	7	12	15	13	15
6	0	2	1	3	8	10	9	11	4	6	5	7	12	14	13	15
5	0	3	0	3	12	15	12	15	0	3	0	3	12	15	12	15
4	0	2	0	2	8	10	8	10	0	2	0	2	8	10	8	10
3	0	1	1	1	4	5	5	5	4	5	5	5	4	5	5	5
2	0	0	1	1	0	0	1	1	4	4	5	5	4	4	5	5
1	0	1	0	1	4	5	4	5	0	1	0	1	4	5	4	5
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

4.1. Reaction tables

Table 4.4: Transposed Canonical folding, 11 self-replications, 132 replications, 113 new string reactions.

15	0	3	3	3	12	15	15	15	12	15	15	15	12	15	15	15
14	0	2	3	3	8	10	11	11	12	14	15	15	12	14	15	15
13	0	1	3	3	4	5	7	7	12	13	15	15	12	13	15	15
12	0	0	3	3	0	0	3	3	12	12	15	15	12	12	15	15
11	0	3	2	3	12	15	14	15	8	11	10	11	12	15	14	15
10	0	2	2	2	8	10	10	10	8	10	10	10	8	10	10	10
9	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	0	0	2	2	0	0	2	2	8	8	10	10	8	8	10	10
7	0	3	1	3	12	15	13	15	4	7	5	7	12	15	13	15
6	0	2	1	3	8	10	9	11	4	6	5	7	12	14	13	15
5	0	1	1	1	4	5	5	5	4	5	5	5	4	5	5	5
4	0	0	1	1	0	0	1	1	4	4	5	5	4	4	5	5
3	0	3	0	3	12	15	12	15	0	3	0	3	12	15	12	15
2	0	2	0	2	8	10	8	10	0	2	0	2	8	10	8	10
1	0	1	0	1	4	5	4	5	0	1	0	1	4	5	4	5
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

4.1. Reaction tables

Table 4.5: Topological folding, 5 self-replications, 106 replications, 145 new string reactions.

15	0	3	3	3	12	15	15	15	12	15	15	15	12	15	15	15
14	0	2	3	3	8	10	11	11	12	14	15	15	12	14	15	15
13	0	3	2	3	12	15	14	15	8	11	10	11	12	15	14	15
12	0	2	2	2	8	10	10	10	8	10	10	10	8	10	10	10
11	0	3	1	3	12	15	13	15	4	7	5	7	12	15	13	15
10	0	2	1	3	8	10	9	11	4	6	5	7	12	14	13	15
9	0	3	0	3	12	15	12	15	0	3	0	3	12	15	12	15
8	0	2	0	2	8	10	8	10	0	2	0	2	8	10	8	10
7	0	1	3	3	4	5	7	7	12	13	15	15	12	13	15	15
6	0	0	3	3	0	0	3	3	12	12	15	15	12	12	15	15
5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	0	0	2	2	0	0	2	2	8	8	10	10	8	8	10	10
3	0	1	1	1	4	5	5	5	4	5	5	5	4	5	5	5
2	0	0	1	1	0	0	1	1	4	4	5	5	4	4	5	5
1	0	1	0	1	4	5	4	5	0	1	0	1	4	5	4	5
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

4.1. Reaction tables

Table 4.6: Transposed Topological folding, 7 self-replications, 110 replications, 139 new string reactions.

15	0	3	3	3	12	15	15	15	12	15	15	15	12	15	15	15
14	0	2	3	3	8	10	11	11	12	14	15	15	12	14	15	15
13	0	1	3	3	4	5	7	7	12	13	15	15	12	13	15	15
12	0	0	3	3	0	0	3	3	12	12	15	15	12	12	15	15
11	0	3	1	3	12	15	13	15	4	7	5	7	12	15	13	15
10	0	2	1	3	8	10	9	11	4	6	5	7	12	14	13	15
9	0	1	1	1	4	5	5	5	4	5	5	5	4	5	5	5
8	0	0	1	1	0	0	1	1	4	4	5	5	4	4	5	5
7	0	3	2	3	12	15	14	15	8	11	10	11	12	15	14	15
6	0	2	2	2	8	10	10	10	8	10	10	10	8	10	10	10
5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	0	0	2	2	0	0	2	2	8	8	10	10	8	8	10	10
3	0	3	0	3	12	15	12	15	0	3	0	3	12	15	12	15
2	0	2	0	2	8	10	8	10	0	2	0	2	8	10	8	10
1	0	1	0	1	4	5	4	5	0	1	0	1	4	5	4	5
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

4.2 Simple Homogeneous Systems

In the simplest homogeneous systems nothing observable ever happens. This is because the only reaction types that occur are self-replications, where $s_x \oplus s_x = s_x$. For instance, in a population of S_1 strings the only string to ever be produced will be S_1 , creating a constant state where strings are continuously lost but immediately replaced by a member of the same species of string [1, 2, 6].

We therefore began our experiments with strings of the new string generation reaction type $s_x \oplus s_x = s_y$. This ensures dynamic populations that will not remain at their initial state. There are a total of 36 reactions of this type across the four reaction tables. Reactions that initially produce the destructor or exploiter strings are dead-end systems, where all the initial strings are eventually replaced by invasive strings. Indeed many reactions of this type meet the same fate, where the initial population is entirely converted to another species.

Table 4.7: String production for reaction type $s_x \oplus s_x = s_y$. Dynamic systems and systems with transitional states are shown in black, and dead-end reactions are shown in grey.

Fold	String
F_{can}	$S_2, S_3, S_4, S_5, S_6, S_7, S_{10}, S_{11}, S_{12}, S_{13}, S_{14}$
F_{t-can}	$S_2, S_4, S_6, S_7, S_{14}$
F_{top}	$S_2, S_3, S_4, S_6, S_8, S_9, S_{10}, S_{11}, S_{12}, S_{13}, S_{14}$
F_{t-top}	$S_2, S_4, S_6, S_7, S_8, S_9, S_{10}, S_{11}, S_{14}$

For this set of experiments we chose to examine S_6, S_7, S_9 and S_{11} , with each system consistently using one type of folding. S_7 is the only string in the group selected that was an initial dead end system, but both S_7 and S_{11} eventually end up as exploiter systems, with S_7 directly declining into S_{15} ,

4.2. Simple Homogeneous Systems

and S_{11} becoming an exploiter system through an intermediate string. S_9 formed a metabolism similar to an exploiter system, with one species slowly being consumed and replaced by another; in another section S_9 formed an opportunistic system where the last remaining strings evenly competed for space and survival. S_6 formed systems that were either structured or opportunistic, the former displayed traits of cooperation and the latter were systems where strings had an equal probability of success, failure, or stagnation.

In most systems where the population is initialised with only one string species, the species concentrations will tend towards the same state, regardless of how many times the simulation is run. Therefore, even though these systems are stochastic, they will appear to be deterministic. However, in certain systems, the true stochastic nature of the model becomes apparent. In these systems the initial phase of the simulation will always be the same, but the end states of these systems are truly stochastic, and they may not render the same results when rerun. These results are described in detail below.

0111 canonical folding

S_7 [0111] is an initial dead-end system where the initial string population becomes overtaken by the exploiter. Figs. 4.1.a and 4.1.b show two systems each initialised with S_7 , Fig. 4.1.a was initialised with a population (M) of 10^3 strings and Fig. 4.1.b was initialised with 10^5 strings. Both systems were only allowed to fold according to the canonical folding type, and both reactions were run for 10 generations. Binary string systems are measured in generations, where a generation is an amount of iterations equal to the number of objects in the population. In a population of 10^3 strings, one

4.2. Simple Homogeneous Systems

generation is equal to 10^3 iterations.

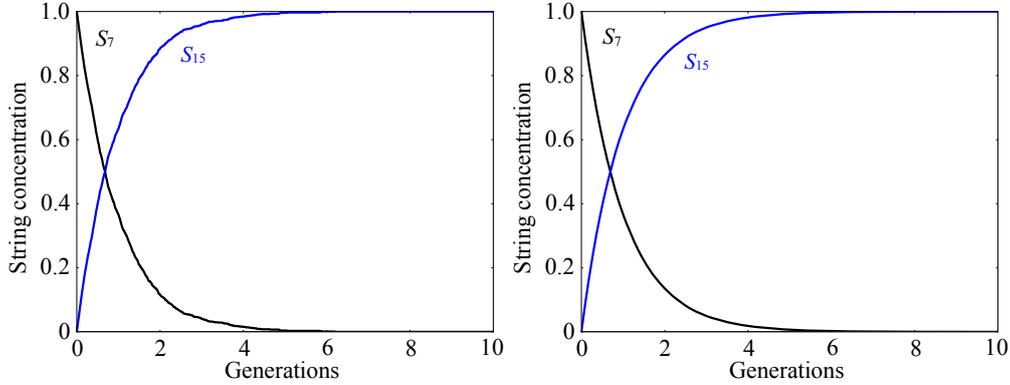


Figure 4.1: (a) left, S_7 - F_{can} , $M = 10^3$, 10 generations. (b) right, S_7 - F_{can} , $M = 10^5$, 10 generations.

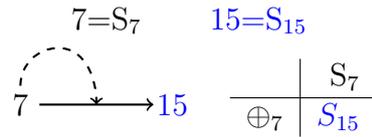
When the interaction algorithm is started two members from the population are randomly selected, but S_7 is the only species in the initial soup and two S_7 strings are thus selected. The two strings interact via the algorithm: one string is converted to an operator that systematically acts on the second string to produce a third string.

$$s_7 \oplus s_7 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 * 1 + 1 * 1 \\ 1 * 1 + 0 * 1 \\ 1 * 1 + 1 * 0 \\ 1 * 1 + 0 * 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = [1111] = s_{15}$$

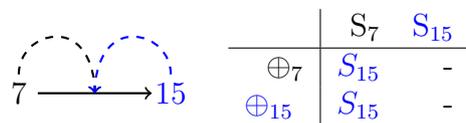
The metabolism of simpler systems can be visualised using lines to represent interactions between strings. A solid line pointing from string a to string b means that string a is being operated on and string b is being produced. A dotted line will span from string x to the centre of a solid line, implying that string x is operating on the string at the beginning of the solid line and producing the string the solid arrow is pointing towards. The table on the right of the metabolic diagram represents the same reaction

4.2. Simple Homogeneous Systems

taking place: the left column houses the operators and the top row contains the string acted upon. This system began with S_7 operating on itself to produce S_{15} :



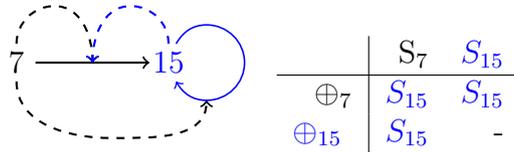
In the first few iterations this will be the only reaction to occur. This is because the concentration of S_{15} is too low for it to be selected as an operator or a string. The probability of S_{15} being selected is proportional to twice its concentration: in Fig. 4.1.a, after the first iteration, there will be one S_{15} in the population of 10^3 strings, meaning that in the second iteration the chance of S_{15} being the first selected string will be $1/1000$ and the chance of S_{15} being the second string selected will be $1/999$, giving a total chance of $2/1000$, or 0.2% . In Fig. 4.1.b this chance will be even less because the population is larger: $1/100000 + 1/99999 \approx 2/100000$ or $2 * 10^{-3}$. The next possible reaction to occur must contain S_7 and S_{15} , and either could be the operator or the string. If the next reaction to occur selected S_{15} as the operator and S_7 as the selected string then the metabolic diagram and table will change in the following way, with S_{15} operating on S_7 and producing S_{15} :



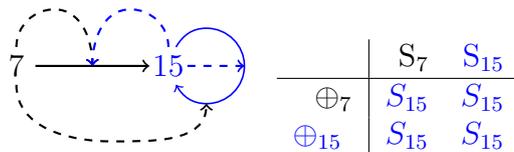
In the metabolic tables “-” is used to indicate that the reaction represented does not yet occur in the metabolic diagram. The next reaction to occur would be the inverse of the previous reaction, with S_{15} selected as the

4.2. Simple Homogeneous Systems

string and S_7 as the selected operator, with this reaction also producing S_{15} .



These are the main two reactions that occur after S_7 has created substantial amounts of S_{15} . The last reaction in this system occurs when S_{15} operates on itself to produce another S_{15} . All four of the reactions in this system lead to the production of S_{15} and the consumption of S_7 , and from the beginning the S_7 strings are quickly metabolised into S_{15} , which soon overwhelms and replaces the initial S_7 population.



From Figs. 4.1.a–4.1.b it can be seen that the S_7 population declines so rapidly that it occupies 50% of the population before half a generation has passed. If S_7 was constantly metabolised from the beginning of the simulation then one S_7 would be replaced by one S_{15} every iteration, yielding a linear decline in S_7 population, where after 500 iterations 500 S_7 strings are replaced, and after one generation, 10^3 iterations, all S_7 strings from the initial population are replaced. The rapid decline is a consequence of the random removal of a string to keep the population constant. In the early stages of the system before S_{15} reaches 50% its low concentration gives S_{15} the advantage of being less likely to be removed by random selection than S_7 . This, in combination with S_{15} being the only string produced, gives

4.2. Simple Homogeneous Systems

S_{15} the ability to initially increase faster than a population where operators transform the string operated on, as opposed to copying and replacing it with a random string, and faster than a system of linear decline, where S_7 is metabolised at a rate of one per iteration.

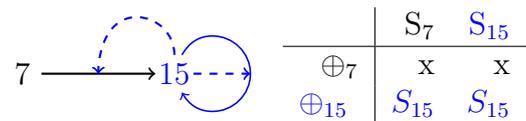
After two generations the S_7 population has dropped to just above 10% of the total population. When S_7 reaches 10% the probability of it being selected to take part in a reaction is equal to the sum of the chances of it occurring in reaction. There are four reactions, but S_7 only occurs in three as the fourth reaction is S_{15} operating on S_{15} :

$$\begin{aligned}
 S_7 \oplus S_7 &: \frac{100}{1000} \times \frac{99}{999} = 0.99\% \approx 1\% \\
 S_7 \oplus S_{15} &: \frac{100}{1000} \times \frac{900}{999} = 9.01\% \approx 9\% \\
 S_{15} \oplus S_7 &: \frac{900}{1000} \times \frac{100}{999} = 9.01\% \approx 9\% \\
 S_{15} \oplus S_{15} &: \frac{900}{1000} \times \frac{899}{999} = 80.99\% \approx 81\%
 \end{aligned}$$

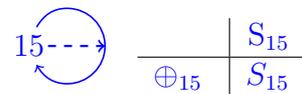
The sum of the three reactions S_7 occurs in is 19%: S_7 has 18% chance to be selected to take part in a reaction as either an operator or a string, and 1% chance to take part as both. S_{15} has a combined chance of 99% of occurring in a reaction, as it has an 81% chance to react with itself and an 18% chance to react with S_7 . So at 10% of the population S_7 has an 18% chance of participating in a reaction and a 10% chance of being removed, and because strings are not metabolised in reactions being randomly removed is the only way to be completely removed from the population. The slow decline in the S_7 population after generation two can be attributed to random removal: the probability of being selected for removal is directly proportional to the concentration of S_7 ; as the population declines the re-

4.2. Simple Homogeneous Systems

removal probability declines as well. An “x” in the metabolic table represents a reaction that is unlikely to occur because one member involved in the reaction has a concentration so low that it is unlikely to be selected. In the diagram below S_7 has such a low concentration that it is unlikely to act as an operator on itself or on S_{15} .



In Fig. 4.1.a the population of S_7 reaches zero just after the sixth generation, in Fig. 4.1.b the S_7 concentration reaches zero just before the eighth generation. The discrepancy in S_7 end points can be attributed to the different population sizes: S_7 will be randomly removed from a system with 10^3 population faster than from a system with 10^5 population.



After the removal of the last S_7 string the population is, once again, homogeneous. Except that this time the population is completely filled with S_{15} strings. Although nothing appears to be happening at this stage S_{15} is actually continually operating on itself, creating a population that appears static while it is actually constantly renewing itself.

1001 transpose topological folding

S_9 [1001] is a system of gradual decline where the initial string population is slowly transformed into another string species. Figs. 4.2.a and 4.2.b show two systems each initialised with S_9 , Fig. 4.2.a was initialised with a population of 10^3 strings and Fig. 4.2.b was initialised with 10^5 strings. Both

4.2. Simple Homogeneous Systems

systems were only allowed to fold according to the transpose topological folding type, and both reactions were run for 10 generations.

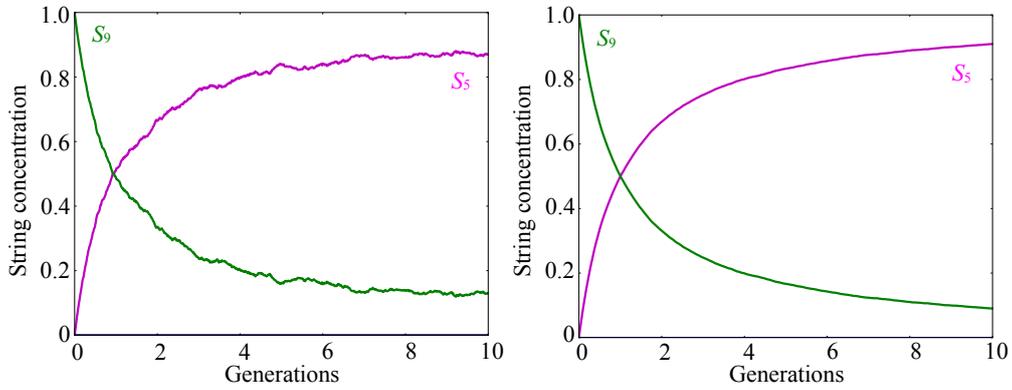
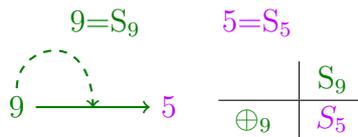


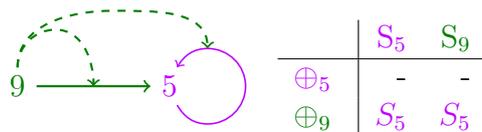
Figure 4.2: (a) left, $S_9-F_{t,top}$, $M = 10^3$, 10 generations. (b) right, $S_9-F_{t,top}$, $M = 10^5$, 10 generations.

This system began with S_9 operating on another S_9 to produce S_5 :

$$s_9 \oplus s_9 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 * 1 + 1 * 0 \\ 0 * 1 + 0 * 0 \\ 1 * 0 + 1 * 1 \\ 0 * 0 + 0 * 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = [0101] = s_5$$

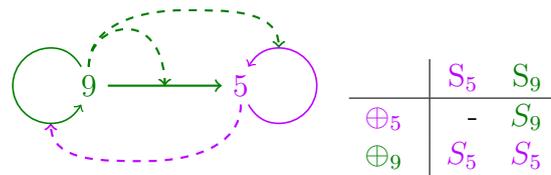


This reaction produces S_5 until S_5 is selected to participate in a reaction. There is an equal chance of S_5 being selected as an operator or as a string. When S_9 operates on S_5 it will produce another S_5 string:

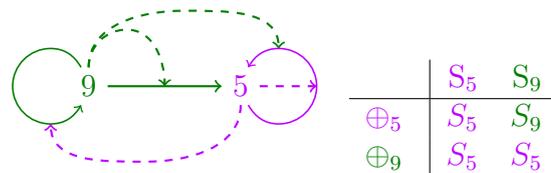


4.2. Simple Homogeneous Systems

When S_5 reaches 50% concentration there is a 25% chance that, if selected, it will operate on S_9 to produce S_9 . This is also the only reaction in this system that produces S_9 . At this stage there are two reactions producing S_5 , both with S_9 as an operator, and one reaction producing S_9 , with S_5 as an operator.



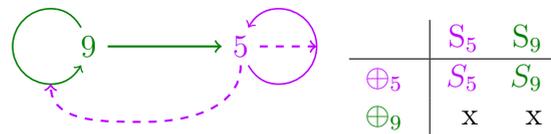
If, at this stage, S_5 were to operate on itself and produce S_9 then this system would reach a steady state where S_5 and S_9 concentrations remain around 50%. This is because as a species concentration rises the chances of it operating on itself and producing the other string increase as well. This is, however, not the case in this system, as S_5 will operate on and produce itself in a self-replication reaction. There are actually no subsets where this symbiosis normally occurs, where each operator produces only the conjugate string species, but these systems can be created through manipulating folding patterns.



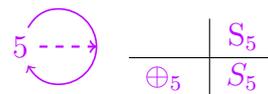
When both species are 50% all reactions have an equal chance of occurring, and 75% of the reactions produce S_5 . From this point the S_9 concentration gradually declines, as there are more reactions producing S_5 than S_9 . The initial population declines much slower than in the S_7-F_{can} system because one of reactions is producing the initial string. As the S_5

4.2. Simple Homogeneous Systems

concentration increases the chance of S_5 operating on S_9 increases as well. When S_5 reaches 90% there is 91% chance of a reaction producing S_5 and 9% for producing S_9 , and there is a 10% chance that S_9 will be removed randomly. This means that after 10 S_9 strings have been randomly removed approximately 9 S_9 strings have been produced by S_5 operating on S_9 . This makes the concentration of S_9 decrease slowly with lower concentrations, and the probability of S_9 being removed and being produced tend towards each other as the S_9 concentration decreases.



In Fig. 4.2.a after 10 generations the S_9 concentration is above 10%, in Fig. 4.2.b the S_9 concentration is less than 10%, and in Figs. 4.1.a and 4.1.b the S_7 is 0% by generation 10. Eventually S_9 will be completely removed from the population by random removal, although this will take many generations because of S_5 being able to produce more S_9 , the probability always being just less than the S_9 concentration.



Once S_9 has been completely removed S_5 operating on S_5 is the only remaining reaction taking place and the systems remain constant with S_5 at 100% of the population.

$$s_5 \oplus s_5 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 * 1 + 0 * 0 \\ 0 * 1 + 1 * 0 \\ 1 * 1 + 0 * 0 \\ 0 * 1 + 1 * 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = [0101] = s_5$$

0110 canonical folding

S_6 [0110] forms a cooperative system with S_9 where both species contribute towards keeping each other stable. Figs. 4.3.a and 4.3.b show two systems each initialised with S_6 , Fig. 4.1.a had an initial population of 10^3 strings and Fig. 4.1.b was initialised with 10^5 strings. Both systems were only allowed to use the canonical folding type, and both were run for 10 generations.

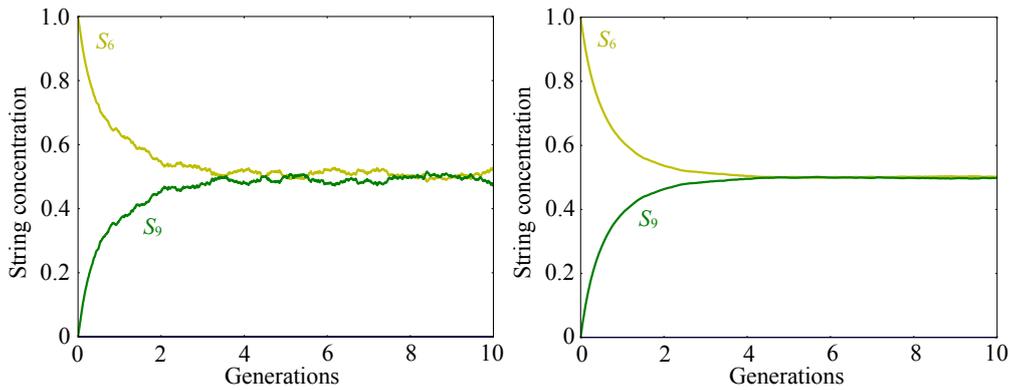
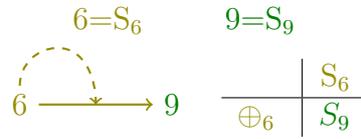


Figure 4.3: (a) left, S_6 - F_{can} , $M = 10^3$, 10 generations. (b) right, S_6 - F_{can} , $M = 10^5$, 10 generations.

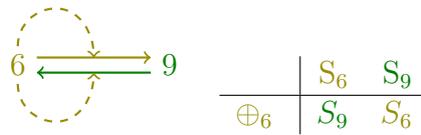
When the system is started S_6 will be selected and will operate canonically on another S_6 to produce S_9 . When S_6 and S_9 fold canonically they form operators that are diagonally symmetrical and inverse of each other.

$$s_6 \oplus s_6 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 * 0 + 1 * 1 \\ 1 * 0 + 0 * 1 \\ 0 * 1 + 1 * 0 \\ 1 * 1 + 0 * 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = [1001] = s_9$$

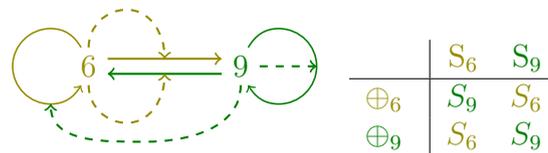
4.2. Simple Homogeneous Systems



Initially S_6 operating on itself is the only reaction taking place: this quickly produces large amounts of S_9 and within one generation the S_9 population has already reached 40%. By this stage S_9 has already been incorporated into reactions as an operator or string. When S_6 operates on S_9 it will produce an S_6 string, contributing to the S_6 population and slowing the rate of decline of S_6 . If, in this system, only S_6 was allowed to operate, it would still reach the same end state as a system where S_9 is allowed to operate as well. This would only happen if the strings produced by an S_9 were destructor strings, which would make the reactions elastic.



However, the reactions where S_9 is an operator do not produce destructor strings. Where S_6 acts on a string and produces the conjugate string, S_9 will produce the string it operated on. This produces a reaction table where both strings have an equal chance of being produced. If S_9 also produced conjugate strings then the reaction table and metabolic diagram would be different, but the resulting end state of the system would remain the same.



The metabolic diagram clearly represents the role of each string in this system: S_6 is responsible for new string generation and always produces the

complimentary partner of the string it operates on, and S_9 will always produce the string it has operated on. Although these are both replication class reactions, reactions of the S_6 type tend to make a population more stable: if only S_6 were allowed operate the species with the highest concentration will be operated on and the conjugate string will be produced, lowering the concentration of the dominating string. If only S_9 were allowed to operate the species with the highest concentration will be operated on and will be replicated, increasing the concentration of the dominating string. If both strings operate in the S_6 way the population will be extremely stable, and if both strings operate in the S_9 way then the population will be extremely stochastic.

1011 topological folding

The S_{11} [1011] string system becomes an exploiter system through production of the intermediate string S_7 . The initial S_{11} population is metabolised into S_7 , which then reacts with S_{11} to produce S_{15} . Figs. 4.4.a and 4.4.b show two systems each initialised with S_{11} , Fig. 4.4.a was initialised with a population of 10^3 strings and Fig. 4.4.b was initialised with 10^5 strings. Both systems were only allowed to fold according to the topological folding type, and both reactions were run for 10 generations.

S_{11} will produce S_7 when it topologically operates on itself. S_{11} and S_7 have structural similarity: both strings are part of set 3 in Fig. 4.2, the group whose members contain one zero bit and three one bits. Having three one bits causes members of this subset to have a high probability of producing the exploiter string during the self-replication reaction under any type of folding. The conditions in the last row of the S_{11} self-reaction $[1 * 0 + 0 * 1]$ are one of the rare instances of a set 3 member self-reacting

4.2. Simple Homogeneous Systems

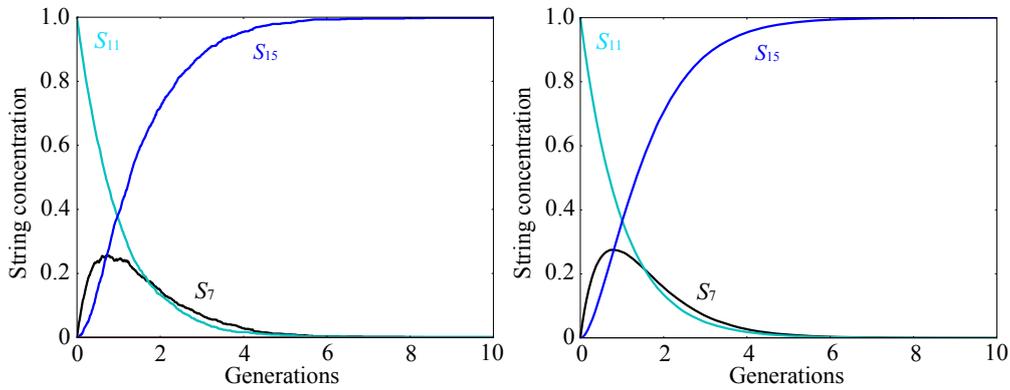
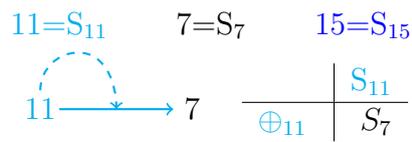


Figure 4.4: (a) left, S_{11} - F_{top} , $M = 10^3$, 10 generations. (b) right, S_{11} - F_{top} , $M = 10^5$, 10 generations.

and not producing an exploiter string.

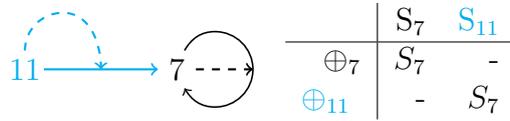
$$s_{11} \oplus s_{11} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 * 1 + 1 * 1 \\ 1 * 1 + 0 * 1 \\ 1 * 0 + 1 * 1 \\ 1 * 0 + 0 * 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} = [0111] = s_7$$

In Figs. 4.4.a and 4.4.b the concentration of S_{11} rapidly declines because of production of large amounts of S_7 ; no reactions replenish the S_{11} population.

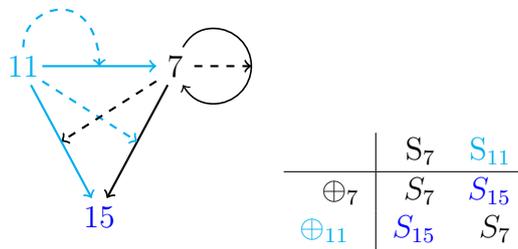


After one generation the S_{11} population is already below 50% concentration. The S_7 species initially increases quickly to above 25% of the population, but reaches its maximum point before one generation has past. S_7 is a self replicator when it folds topologically, and while this aids the S_7 population in growth it is, in this case, not sufficient for a sustained S_7 population.

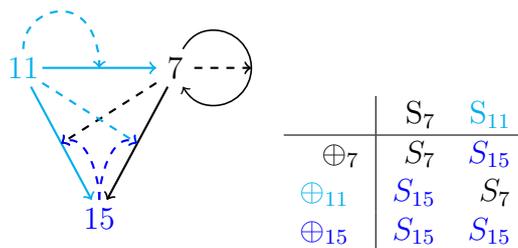
4.2. Simple Homogeneous Systems



The self-replicating ability of S_7 in this case is not enough for sustained population because S_7 does not create any mutually beneficial reaction cycles. When S_{11} and S_7 interact ($7 \oplus 11 \vee 11 \oplus 7$) they produce the exploiter string, S_{15} . No reactions between S_{11} and S_7 produce S_{11} , and no interactions with the exploiter string produce S_{11} .

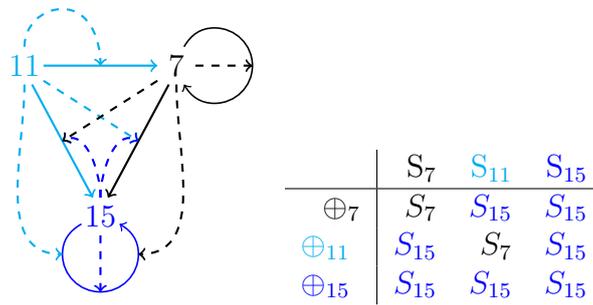


In Figs. 4.4.a and 4.4.b the interaction between S_{11} and S_7 is evident as the S_{15} population soon achieves and maintains the same growth rate the S_7 population initially had. S_{15} will operate on both S_{11} and S_7 to produce itself, maintaining the step growth rate.

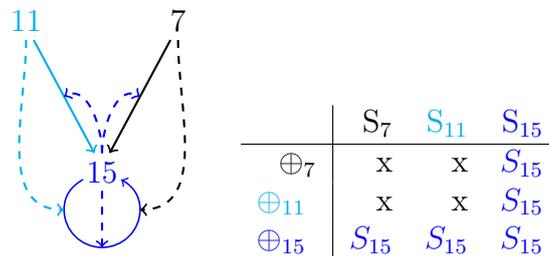


S_{11} and S_7 will produce S_{15} when they operate on it in addition to S_{15} being a self-replicator. Out of nine possible reactions only two produce non-exploiter strings, the self-replication of S_{11} and S_7 both produce S_7 , and all other reactions produce the exploiter string.

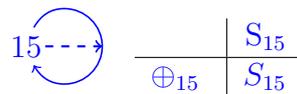
4.2. Simple Homogeneous Systems



In Figs. 4.4.a and 4.4.b in the first generation S_7 decreases at a rate slower than S_{11} ; this is because there are two reactions capable of producing S_7 and no reactions producing S_{11} . After one generation the concentration of S_{11} is lower than the S_7 concentration and both are less than 20% of the population. After two generations the S_{11} and S_7 concentrations are so low that they almost only react with S_{15} , and little S_7 is produced from S_{11} or S_7 self-reactions.



Between generations two and four the S_{11} and S_7 populations diminish completely and their effect on the system becomes negligible. The exploiter string has completely taken over the system by producing itself during its interactions with other strings. Eventually the population reaches its final state when it is entirely S_{15} , and the only reaction occurring is S_{15} self replication.



1001 topological folding

The S_9 [1001] string system becomes a semi-stable system after a transitional phase in which two intermediate strings are produced. The initial population of S_9 strings interact and the population is partially metabolised into S_3 strings.

$$s_9 \oplus s_9 = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 * 1 + 0 * 0 \\ 1 * 1 + 0 * 0 \\ 1 * 0 + 0 * 1 \\ 1 * 0 + 0 * 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = [0011] = s_3$$

The operation of S_3 on itself and S_9 produces S_1 and S_5 respectively. Further operation of S_9 on S_5 produces the last new string of the system, the exploiter string S_{15} . From here on only partial reaction tables will be shown, and not metabolic diagrams. Metabolic diagrams become increasingly complex as the number of different species in the population increases, as number of reactions between objects is exponentially proportional to the number of species present.

Figs. 4.5.a and 4.5.b show two systems each initialised with S_9 . Fig. 4.5.a was initialised with a population of 10^3 strings and Fig. 4.5.b was initialised with 10^5 strings. Both systems were only allowed to fold according to the topological folding type, and both systems were run for 10 generations.

Fig. 4.6.a shows the same system as 4.5.a but after 800 generations. Fig. 4.6.b shows the same system as 4.5.b but after 10^3 generations.

In Figs. 4.5.a and 4.5.b the S_9 strings in the initial population interact and S_3 is produced at a high rate. During the first generation the S_3 population occupies almost 40% of the reactor in both systems.

S_9 will operate on S_3 to produce another S_3 , assisting the initial rate of

4.2. Simple Homogeneous Systems

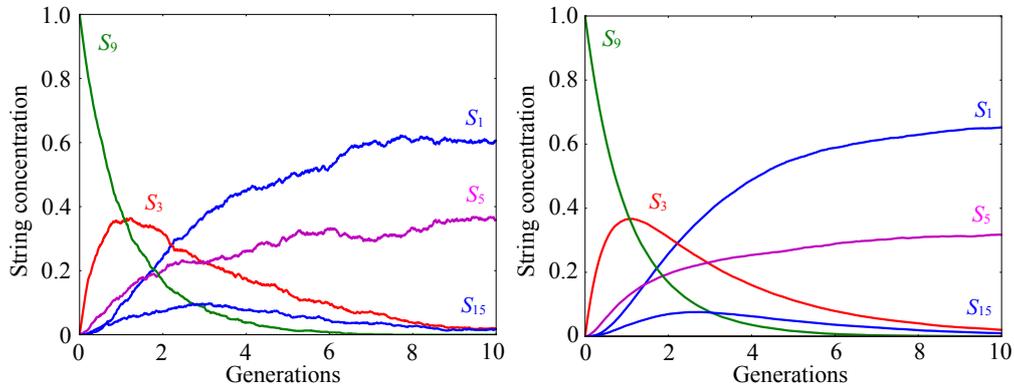


Figure 4.5: (a) left, S_9-F_{top} , $M = 10^3$, 10 generations. (b) right, S_9-F_{top} , $M = 10^5$, 10 generations.

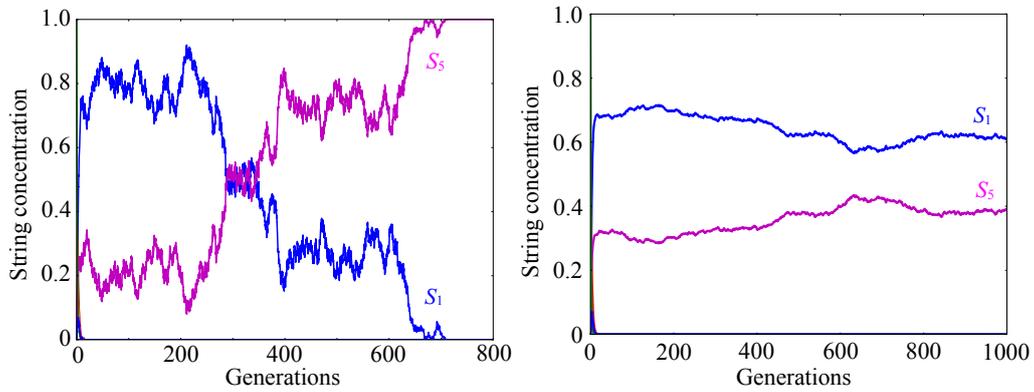


Figure 4.6: (a) left, S_9-F_{top} , $M = 10^3$, 800 generations. (b) right, S_9-F_{top} , $M = 10^5$, 1000 generations.

	S_9
\oplus_9	S_3

S_3 production. When S_3 operates on itself and S_9 two new strings, S_1 and S_5 , are produced. S_1 is the result of the S_3 self-reaction, and S_5 is produced when S_3 operates on S_9 .

	S_3	S_9
\oplus_3	S_1	S_5
\oplus_9	S_3	S_3

In Figs. 4.5.a and 4.5.b, in the first generation, S_1 and S_5 are the first

4.2. Simple Homogeneous Systems

two strings that appear after the sharp rise of S_3 . The addition of these strings to the population changes the dynamics of the system, resulting an increased production of S_1 and S_5 . In the second generation the S_9 population is below 20% and the S_3 concentration has passed its apex and has started declining. The low S_9 concentration is the reason for the decline of S_3 , as S_9 was the main producer of S_3 . The addition of S_1 and S_5 to the population shifts the system towards producing S_1 and S_5 . S_3 is produced in two more reactions, $S_5 \oplus S_3$ and $S_9 \oplus S_1$, and S_9 is produced when $S_5 \oplus S_9$. The exploiter string, S_{15} , is also produced when $S_9 \oplus S_5$. Strings and operators have their own characteristic properties, S_1 and S_5 stand out as strings that will often produce themselves when operated on, S_1 will mostly produce itself as an operator, and S_5 will produce the same string it operates on.

	S_1	S_3	S_5	S_9
\oplus_1	S_1	S_1	S_5	S_1
\oplus_3	S_1	S_1	S_5	S_5
\oplus_5	S_1	S_3	S_5	S_9
\oplus_9	S_3	S_3	S_{15}	S_3

S_{15} is the last new string generated by this system. Although S_{15} usually tends to exploit a system, in this system it reaches its maximum concentration between generations two and four.

	S_1	S_3	S_5	S_9	S_{15}
\oplus_1	S_1	S_1	S_5	S_1	S_5
\oplus_3	S_1	S_1	S_5	S_5	S_5
\oplus_5	S_1	S_3	S_5	S_9	S_{15}
\oplus_9	S_3	S_3	S_{15}	S_3	S_{15}
\oplus_{15}	S_3	S_3	S_{15}	S_{15}	S_{15}

 4.2. Simple Homogeneous Systems

By the fourth generation S_9 occupies less than five percent of the population, and only one reaction produces S_9 . The reaction producing S_9 depends on the S_9 concentration, so S_9 has less chance of replication as its concentration decreases. If S_9 was produced by a dominant pair of strings then it may have become constant at a lower concentration as opposed to diminishing completely.

The next pair of strings that disappear from the system are S_3 and S_{15} . S_3 is produced by operators S_5 , S_9 and S_{15} : the S_9 population depleted quickly, S_5 only produces S_3 when it acts on S_3 , and S_{15} operates on S_1 or S_3 to produce S_3 . Besides the initial reaction creating S_{15} , $(S_9 \oplus S_5)$, it is created when S_{15} operates on S_5 , S_9 , or S_{15} , and when S_{15} is operated on by S_5 , S_9 , or S_{15} . The reliance of S_3 and S_{15} on S_9 and S_{15} as operators led to their demise.

	S_1	S_3	S_5	S_{15}
\oplus_1	S_1	S_1	S_5	S_5
\oplus_3	S_1	S_1	S_5	S_5
\oplus_5	S_1	S_3	S_5	S_{15}
\oplus_{15}	S_3	S_3	S_{15}	S_{15}

S_3 and S_{15} could have formed a cooperative network if the S_3 operations on S_3 and S_{15} created S_3 and S_{15} , in either order. In scenario one the S_3 self-reaction produces S_{15} and $S_3 \oplus S_{15}$ produces S_3 . This will form a stable cooperative network as S_6 and S_9 did in the S_6 topological system. In the second scenario $S_3 \oplus S_3$ produces S_3 and $S_{15} \oplus S_{15}$ produces S_{15} . This reaction network is not stable, and this is the type of cooperation that S_1 has with S_5 .

In the end it was this underlying reaction network that allowed S_1 and S_5 to thrive past the disappearance of their counterparts. In this unstable

4.2. Simple Homogeneous Systems

reaction network the operators will produce whatever they act on, leaving the production chance for each string at its concentration e.g. if the S_1 population is 60%, there is a 60% chance of S_1 being produced and/or randomly removed, and the same is true for S_5 .

	S_1	S_5
\oplus_1	S_1	S_5
\oplus_5	S_1	S_5

Figs. 4.6.a and 4.6.b show how stochastic this type of reaction network is over very long periods. While the system in Fig. 4.5.b may appear very stable, Fig. 4.5.a shows how unstable this system can become. Fig. 4.6.b also illustrates the stability of larger systems, as after 800 generations the cooperative pair in Fig. 4.6.a have gone through much more turbulence than the network pair in the larger reactor. However, even when compared with Figs. 4.3.a and 4.3.b, the instability of the network in the large reactor in Fig. 4.6.b is apparent, and the destiny of the strings in such reaction networks is purely up to chance.

0110 transpose topological folding

In the S_6 [0110] transpose topological system eventually becomes dominated by a stable co-operative string pair. The initial string population is metabolised into S_{10} , which operates on itself and S_6 to produce S_5 and S_9 respectively. The result is an interesting reaction network in which the dominant cooperative string-pair maintain themselves and the weaker pair, while the weaker pair exclusively produce the dominant members. Figs. 4.7.a and 4.7.b show two systems each initialised with S_6 , Fig. 4.7.a was initialised with a population of 10^3 strings and Fig. 4.7.b was initialised

4.2. Simple Homogeneous Systems

with 10^5 strings. Both systems were only allowed to fold according to the transpose topological folding type, and both reactions were run for 10 generations.

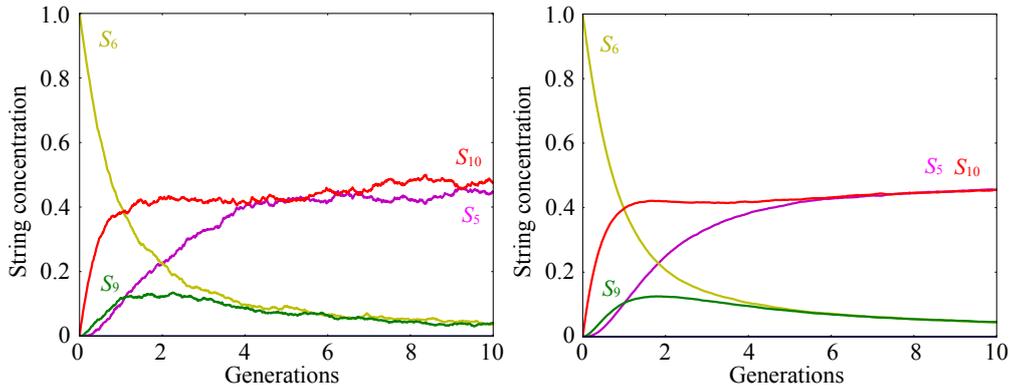


Figure 4.7: (a) left, S_6 - F_{t-top} , $M = 10^3$, 10 generations. (b) right, S_6 - F_{t-top} , $M = 10^5$, 10 generations.

When the reactor is started reactions between S_6 strings and transpose topologically folded S_6 operators will occur, producing S_{10} strings.

$$s_6 \oplus s_6 = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 * 0 + 0 * 1 \\ 1 * 0 + 1 * 1 \\ 0 * 1 + 0 * 0 \\ 1 * 1 + 1 * 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = [1010] = s_{10}$$



The interactions that occur between S_6 and S_{10} , specifically S_{10} operating on S_6 and itself, introduce new strings, S_9 and S_5 , into the system. In Figs. 4.7.a and 4.7.b, by the end of the first generation, the S_{10} concentration has rapidly increased to above 40% and has surpassed the S_6 concentration.

4.2. Simple Homogeneous Systems

	S_6	S_{10}
\oplus_6	S_{10}	S_{10}
\oplus_{10}	S_9	S_5

During the first generation S_9 and S_5 appear in the system. While the S_9 concentration initially increases faster than S_5 , the S_5 population reaches its apex before the second generation, and after that the S_9 population begins a slow decline until merging with the S_6 population. The S_5 population begins growth at a slower rate than the S_9 population, but the growth is maintained until S_5 merges with the S_{10} population.

The subset of strings in this system form a very interesting reaction network. The S_6 operator will produce S_{10} strings regardless of the string it operates on, and in the same manner the S_9 operator will only produce S_5 strings. The S_5 operator will produce the same string it acts on, so the probability of each string being produced is its equal to its concentration. S_{10} is the interesting operator in this system, as it produces strings that are different but structurally related to the string operated on. As read from the reaction table the sequence is the same as the S_5 sequence backwards. With the current string production of the other three operators S_{10} makes the perfect operator to balance the system: S_5 is stochastically keeping everything constant while S_6 and S_9 are respectively producing S_{10} and S_5 , S_{10} acts as a dampening operator assisting each string in reaching the concentration of its structural counterpart. Whenever S_{10} operates S_5 and S_{10} will strive towards each other, while S_6 and S_9 will tend towards each other. If only S_{10} were allowed to operate the concentrations of the two pairs would begin migrating towards each other until $S_5 = S_{10}$ and $S_6 = S_9$ and they would remain at the concentration they met.

Figs. 4.7.a and 4.7.b do not show the demise of S_6 and S_9 . They were

4.2. Simple Homogeneous Systems

	S_5	S_6	S_9	S_{10}
\oplus_5	S_5	S_6	S_9	S_{10}
\oplus_6	S_{10}	S_{10}	S_{10}	S_{10}
\oplus_9	S_5	S_5	S_5	S_5
\oplus_{10}	S_{10}	S_9	S_6	S_5

dependant on the strings they produced, S_5 and S_{10} , for replication, but the strings they produced didn't fully return the favour.

	S_5	S_{10}
\oplus_5	S_5	S_{10}
\oplus_{10}	S_{10}	S_5

While S_5 and S_{10} helped S_6 and S_9 self-replicate, they also formed their own closed reaction network. Eventually S_6 and S_9 were out-competed by S_5 and S_{10} , who formed a stable cooperative network. This network had a similar structure to the S_6 canonical system, where one string is produced by both self-replications, and the other string is produced by the pair reactions. In this instance S_5 is produced by the self-replications and S_{10} is produced by the pair of strings reacting.

0110 topological folding

The S_6 [0110] string system eventually becomes semi-stable after a transitional phase where intermediate strings were produced. The initial string population is partially metabolised into S_3 , setting off a chain of reactions that leads to the production of four other strings: S_1 , S_2 , S_4 and S_5 . Figs. 4.8.a and 4.8.b show two systems each initialised with S_6 , Fig. 4.8.a was initialised with a population of 10^3 strings and Fig. 4.8.b was initialised with 10^5 strings. Both systems were only allowed to fold according to the topological folding type, and both systems were run for 10 generations.

4.2. Simple Homogeneous Systems

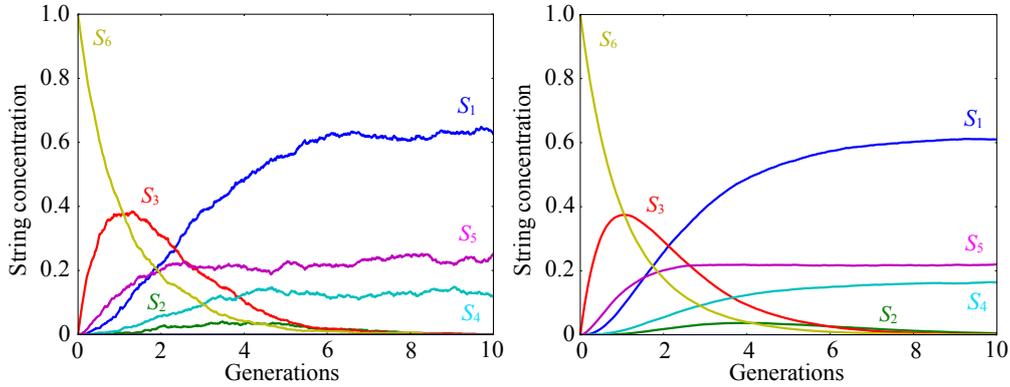


Figure 4.8: (a) left, S_6-F_{top} , $M = 10^3$, 10 generations. (b) right, S_6-F_{top} , $M = 10^5$, 10 generations.

The initial reactions of the system mostly include S_6 operating on other S_6 strings, which produces large quantities of S_3 .

$$s_6 \oplus s_6 = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 * 0 + 1 * 1 \\ 0 * 0 + 1 * 1 \\ 0 * 1 + 1 * 0 \\ 0 * 1 + 1 * 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = [0011] = s_3$$

In Figs. 4.8.a and 4.8.b the S_3 population reaches its peak by the first generation, after reaching almost 40% concentration.

$$\begin{array}{c|c} & S_6 \\ \oplus_6 & S_3 \end{array}$$

The interactions that occur between S_3 and S_6 add new strings to the system, S_1 and S_5 . S_5 is produced by the S_3 self-reaction and S_6 is produced when S_3 operates on S_6 . The initial increased growth rate of S_5 versus S_1 can be attributed to S_5 being produced by S_3 operating on S_6 , as there is initially more S_6 than S_3 to operate on.

By the second generation the S_3 concentration is declining and is slightly above the S_1 concentration that it is about to intersect. S_1 exploits S_3

4.2. Simple Homogeneous Systems

	S_3	S_6
\oplus_3	S_1	S_5
\oplus_6	S_3	S_3

in their reaction network, allowing S_1 to maintain a steady growth rate and surpass S_5 . The S_5 population has also gained a foothold and has increased passed S_6 , which has been out-competed by S_1 , S_5 , and S_3 . The introduction of S_1 and S_5 into the reaction network adds one new member, S_4 , and produces an existing member, S_6 .

	S_1	S_3	S_5	S_6
\oplus_1	S_1	S_1	S_5	S_4
\oplus_3	S_1	S_1	S_5	S_5
\oplus_5	S_1	S_3	S_5	S_6
\oplus_6	-	S_3	-	S_3

Initially the only reaction producing S_4 is the operation of S_1 on S_6 , leading to a slow increase in the S_4 concentration. Between generations two and four the S_4 concentration increases past S_3 and S_6 , who are still steadily declining. The S_4 population is sustained by replication reactions involving S_4 as string and S_1 , S_3 or S_5 as operators. The operation of S_4 on S_3 and S_6 also introduces S_2 into the system.

	S_1	S_3	S_4	S_5	S_6
\oplus_1	S_1	S_1	S_4	S_5	S_4
\oplus_3	S_1	S_1	S_4	S_5	S_5
\oplus_4	-	S_2	-	-	S_2
\oplus_5	S_1	S_3	S_4	S_5	S_6
\oplus_6	-	S_3	-	-	S_3

The production of S_2 depends on S_4 and S_5 as operators, and S_2 , S_3 and S_6 as strings. Between generations four and six the S_2 concentration surpasses both S_6 and S_3 , but the dependence of S_2 on itself, S_3 and S_6

4.2. Simple Homogeneous Systems

hinders it from greatly increasing its concentration. S_2 , S_3 and S_6 eventually disappear from the system because of their inability to lock into mutually beneficial reaction cycles.

	S_1	S_2	S_3	S_4	S_5	S_6
\oplus_1	S_1	-	S_1	S_4	S_5	S_4
\oplus_2	-	S_1	S_1	-	-	S_1
\oplus_3	S_1	S_1	S_1	S_4	S_5	S_5
\oplus_4	-	S_2	S_2	-	-	S_2
\oplus_5	S_1	S_2	S_3	S_4	S_5	S_6
\oplus_6	-	S_3	S_3	-	-	S_3

The only reaction S_6 is produced in depends on the S_6 concentration and requires S_5 as an operator. As more S_5 strings are produced the concentration of S_6 decreases less rapidly, but this is not enough to save the S_6 population from depletion.

	S_1	S_2	S_3	S_4	S_5
\oplus_1	S_1	-	S_1	S_4	S_5
\oplus_2	-	S_1	S_1	-	-
\oplus_3	S_1	S_1	S_1	S_4	S_5
\oplus_4	-	S_2	S_2	-	-
\oplus_5	S_1	S_2	S_3	S_4	S_5

S_6 was the main producer of S_3 , and operated on strings S_2 , S_3 and S_6 to produce S_3 , while S_5 also operated on S_3 to produce S_3 . This links the demise of S_6 , the main operator producing S_3 , with the depletion of S_3 . The reliance of S_2 on itself and S_3 as strings also leads to the eventual depletion of S_2 . The S_2 population maintains itself for a short time after the depletion of S_3 by replication reactions with S_4 and S_5 as operators.

Eventually only S_1 , S_4 and S_5 remain. The S_4 operator produces destructor strings when operating on the remaining strings. These reactions are elastic, and the S_4 operator has no effect on the system. The S_1 and

 4.3. Random and Selective Homogeneous Systems

	S_1	S_2	S_4	S_5
\oplus_1	S_1	-	S_4	S_5
\oplus_2	-	S_1	-	-
\oplus_4	-	S_2	-	-
\oplus_5	S_1	S_2	S_4	S_5

S_5 operators produce the same strings they operate on. This forms the same type reaction network that exists in the S_9 topological system. The concentrations of these three strings appear to be relatively constant in Fig. 4.8.b, but Fig. 4.8.a shows how stochastic the system actually is, and Figs. 4.6.a and 4.6.b give an indication of how unstable the system will become hundreds or thousands of generations later. This remaining system is also strangely robust: any one of the three strings may deplete, but the remaining two strings will continue to be produced in the same stochastic manner.

	S_1	S_4	S_5
\oplus_1	S_1	S_4	S_5
\oplus_4	-	-	-
\oplus_5	S_1	S_4	S_5

4.3 Random and Selective Homogeneous Systems

In the following section two homogeneous systems, S_6 and S_9 , are subjected to random and selective folding scenarios. S_6 and S_9 were chosen because a) the self-reactions of S_6 and S_9 are both classified as new string generating reactions in Table 4.7, Section 4.2, and b) their performance in Section 4.2 with regard to potential for string species emergence confirmed their ability as new string generators.

0110 random folding

In the S_6 [0110] random folding string system the initial self-reactions metabolise the S_6 population into S_9 , S_3 , and S_{10} . This sets reactions in motion that lead to the production of seven other strings: S_1 , S_2 , S_4 , S_5 , S_8 , S_{12} and S_{15} . Together these 11 strings form six subgroups, but before the 10th generation the subgroup of S_6 and S_9 has already died out. Over hundreds over generations three of the five subgroups slowly dominate the population. Figs. 4.9.a and 4.9.b show two time frames from the same simulation, each initialised with S_6 and with a population of 10^5 strings. The system was only allowed to utilize random folding.

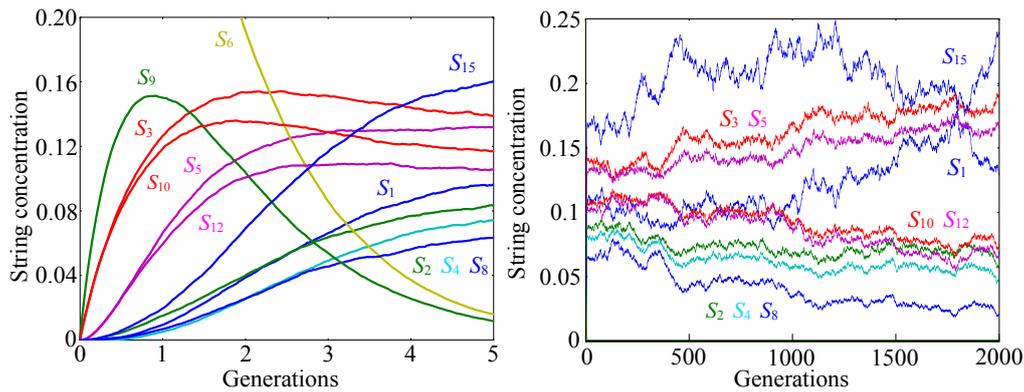


Figure 4.9: (a) left, S_6 - F_{ran} , $M = 10^5$, 25 generations. (b) right, S_6 - F_{ran} , $M = 10^5$, 2000 generations.

The initial self-reactions of S_6 lead to the production of S_9 , S_3 , and S_{10} . S_9 is initially produced at twice the rate of S_3 and S_{10} because there are two S_6 self reactions that produce S_9 , and S_3 and S_{10} are only produced by one reaction each. Random folding will select each folding type in a relatively equal ratio, with longer simulations yielding more averaged folding type selection.

4.3. Random and Selective Homogeneous Systems

	$\oplus_{6\cdot cf}$	$\oplus_{6\cdot tc}$	$\oplus_{6\cdot tp}$	$\oplus_{6\cdot tt}$
S_6	S_9	S_9	S_3	S_{10}

In the following table the operators with their respective folding types are in the top row, and the string being operated on is in the left hand column. In the top column, after the operator species number, are two identifier letters for each folding type: cf (canonical folding), tc (transposed canonical folding), tp (topological folding), and tt (transposed topological folding). The canonical and transposed canonical self-reactions both produce S_9 , while the topological operation produces S_3 and the transposed topological operation produces S_{10} . There is a high probability that the three newly produced strings will encounter S_6 in the string population before they encounter each other. However, there is an equal chance that they will encounter S_6 as either an operator or as a string.

	$\oplus_{6\cdot cf}$	$\oplus_{6\cdot tc}$	$\oplus_{6\cdot tp}$	$\oplus_{6\cdot tt}$
S_6	S_9	S_9	S_3	S_{10}
S_9	S_6	S_6	S_{12}	S_{10}
S_3	S_3	S_3	S_3	S_2
S_{10}	S_5	S_5	S_{15}	S_{10}

The table above shows the strings produced when S_6 , S_9 , S_3 , and S_{10} are operated on by S_6 for each folding method. Their interactions lead to the production of S_2 , S_5 , S_{12} , and S_{15} . The following table contains the strings produced when S_6 is operated on by itself, S_3 , S_9 , and S_{10} for each folding type.

	$S_{6\oplus cf}$	$S_{6\oplus tc}$	$S_{6\oplus tp}$	$S_{6\oplus tt}$
\oplus_6	S_9	S_9	S_3	S_{10}
\oplus_9	S_6	S_6	S_{12}	S_5
\oplus_3	S_5	S_{12}	S_5	S_{12}
\oplus_{10}	S_3	S_{10}	S_9	S_9

4.3. Random and Selective Homogeneous Systems

In both of the above tables S_6 and S_9 are represented adjacent to each other, this is because S_9 is initially produced at a greater rate than S_3 and S_{10} , and because S_9 is structurally related to S_6 . S_3 and S_{10} are also structurally related to S_6 and S_9 , all four of these strings contain two zero bits and two one bits. The operation of S_9 , S_3 , and S_{10} on S_6 lead to the introduction of S_5 and S_{12} , with S_3 exclusively producing S_5 and S_{12} .

	$\oplus_{6\cdot cf}$	$\oplus_{6\cdot tc}$	$\oplus_{6\cdot tp}$	$\oplus_{6\cdot tt}$
S_2	S_1	S_1	S_3	S_2
S_5	S_{10}	S_{10}	S_0	S_{10}
S_{12}	S_{12}	S_{12}	S_{12}	S_8
S_{15}	S_{15}	S_{15}	S_{15}	S_{10}

	$S_{6\oplus cf}$	$S_{6\oplus tc}$	$S_{6\oplus tp}$	$S_{6\oplus tt}$
\oplus_2	S_1	S_8	S_1	S_8
\oplus_5	S_{12}	S_5	S_6	S_6
\oplus_{12}	S_{10}	S_3	S_{10}	S_3
\oplus_{15}	S_{15}	S_{15}	S_{15}	S_{15}

In Fig. 4.9.a after one generation the S_9 population has already peaked and is declining, while red pair, S_3 and S_{10} , and purple pair, S_5 and S_{12} , are still increasing. The next two strings are S_{15} and S_2 , which were only produced once each by the initial reactions of the first four strings. They are followed by S_1 and S_8 , which are produced by S_6 reacting with S_2 . The string at the lowest concentration in generation one is S_4 , which is produced by S_6 operating on S_8 , and S_1 operating on S_6 .

	$\oplus_{6\cdot cf}$	$\oplus_{6\cdot tc}$	$\oplus_{6\cdot tp}$	$\oplus_{6\cdot tt}$
S_1	S_2	S_2	S_0	S_2
S_8	S_4	S_4	S_{12}	S_8
S_4	S_8	S_8	S_0	S_8

4.3. Random and Selective Homogeneous Systems

	$S_{6\oplus cf}$	$S_{6\oplus tc}$	$S_{6\oplus tp}$	$S_{6\oplus tt}$
\oplus_1	S_4	S_4	S_4	S_4
\oplus_8	S_2	S_2	S_8	S_1
\oplus_4	S_8	S_1	S_2	S_2

The reaction tables shown so far only account for reactions where one of the reactants is S_6 , as either a string or as an operator. S_6 is not the only string reacting in the system, but the initially high concentration of S_6 causes it to have many reactions with its own products. It is interesting to note that S_6 produces a closed subset of strings through self reactions, and none of the reactions between the progeny of S_6 add any new strings to the population. The only strings outstanding from this subset are the destructor, whose production is prohibited, and the subset of strings containing S_7 , S_{11} , S_{13} and S_{14} , which are structurally related by all containing three one bits.

In Fig. 4.9.a several groups can be seen differentiating from the start of the simulation. While the S_9 population initially grows faster than any other string sort there are not enough reactions to maintain its concentration, and the effects of out-competition soon become apparent as the S_9 population begins depleting before the end of the first generation. The first two emergent string pairs, S_3/S_{10} , and S_5/S_{12} , each split and begin re-merging towards the fifth generation. It appears as though the red strings (S_3 and S_{10}) are a pair and the purple strings (S_5 and S_{12}) are a pair, although this is not the case, as towards the fifth generation S_3 and S_5 , and S_{10} and S_{12} begin pairing. This is slightly unusual as strings with structural similarity usually support each other (S_3 is related to S_{12} while S_5 is related to S_{10}), but the myriad of other strings and folding types available change the dynamics and increase the plasticity system, allowing the formation of new relationships that may utilize alternative metabolic pathways. The

4.3. Random and Selective Homogeneous Systems

next strings that appear together are S_{15} and S_2 , but the advantage that S_{15} has as an exploiter is apparent as it has overtaken S_2 before the first generation, and by the fifth generation it has surpassed S_3 , S_5 , S_{10} and S_{12} , and it is almost double the S_2 concentration. The last strings to appear are S_1 and S_8 , followed by S_4 . By the fifth generation S_1 has overtaken S_2 , and S_4 has overtaken S_8 to join S_2 . S_6 and S_9 are structurally related and they have mutually beneficial relationships when operating on each other canonically or transposed canonically. However, the only other strings that can produce S_6 and S_9 are S_5 and S_{10} respectively. The lack of beneficial relationships incapacitates S_6 and S_9 , and they are unable to survive for more than 25 generations in S_6 random systems.

In Fig. 4.9.b the strings appear in almost the same order as in Fig. 4.9.a after five generations. In order, and sub-grouped, they are: $[(S_{15})(S_3, S_5)(S_1)](S_{10}, S_{12})(S_2, S_4, S_8)$.

The most dominant group, $[(S_{15})(S_3, S_5)(S_1)]$, actually consists of three subgroups. S_3 and S_5 are structurally related, each containing two one bits. While S_3 and S_5 share mutually beneficial relationships with S_{15} , S_1 is almost only produced by S_3 and is not at all produced by S_{15} , but it is capable of converting S_{15} into S_3 and S_5 .

Even though S_{10} and S_{12} are related to S_3 and S_5 they do not fare as well, and their concentration steadily declines from 400 generations. It was speculated that this was due to S_{10} and S_{12} being less dependant on S_{15} than S_3 and S_5 were, but both pairs share the same summed dependence on S_{15} .

The dependence score yields, as a percentage, how much a particular string contributes, as an operator or a string, to a specific strings production. It is useful when asking ‘at time T in a system how much does string

 4.3. Random and Selective Homogeneous Systems

X influence string Y : for all combinations of operator and string that produce string Y , how many times is X involved, either as an operator or a string, out of the total amount of strings and operators involved? Dependence is also state specific, and dependence values will change if strings enter or leave the system.

In terms of dependence string pair S_3 and S_5 and string pair S_{10} and S_{12} share combined dependence scores on S_{15} of 40.3% per pair, meaning that S_{15} is involved in about 20.15% of reactions involving S_3 , S_5 , S_{10} and S_{12} . S_{15} is equally dependent on S_3 and S_{12} (19.6%) and on S_5 and S_{10} (7.1%), and it is the most dependent on itself of all strings (46.4%).

The dependence tables also revealed that S_{10} and S_{12} are not at all dependent on S_1 , but they are twice as dependent on S_2 , S_4 , and S_8 as S_3 and S_5 are. S_3 and S_5 are on average 13.3% dependent on S_1 , while S_1 is 29.3% dependent on itself and S_3 , 8.6% on S_5 , 3.4% on S_{10} , and not dependent on S_{12} .

Even though S_1 only contributes 13.3% to S_3 and S_5 the fact that it always operates on S_{15} to produce S_5 , and always produces S_3 when operated on by S_{15} , means that S_3 and S_5 will be increased by S_1 when the S_{15} is concentration high. In return S_1 benefits from an increased S_3 concentration. Even though S_{10} and S_{12} have exactly the same dependence on each other and on S_{15} , as S_3 and S_5 do on each other and S_{15} , it is their lack of relationship with a string like S_1 , which reacts with the exploiter to their advantage. S_4 and S_8 interact with S_{15} to produce S_{10} and S_{12} , but not through all reaction tables, and occasionally S_5 is produced when S_{15} is operated on.

Not having a helper string, combined with being more dependent on weaker strings S_2 , S_4 , and S_8 , lead to decreasing S_{10} and S_{12} concentrations.

4.3. Random and Selective Homogeneous Systems

Even though S_1 , S_2 , S_4 , and S_8 are all not dependent on S_{15} , S_1 indirectly benefits from S_{15} through S_3 . S_1 is equally dependent on itself and S_3 (29.3%). S_8 benefits the most from S_{12} (29.3% each) and equally from itself and S_4 (25.9%). S_4 depends equally on itself, S_8 and S_{12} (19.0% each), while S_2 mostly depends on itself (22.4%) and S_3 (19%). The lack of relationship between S_{15} and the subgroup S_2 , S_4 , and S_8 lead to the internal cooperative effects of the group being downplayed. The positive effects of depending on successful strings are evident. S_2 depends more on successful strings than S_4 , and S_4 depends more on successful strings than S_8 . This effect of this can be seen in Fig. 4.9.b, where S_2 , S_4 , and S_8 are tightly grouped for the first four hundred generations, and then they slowly drift apart for the remainder of the simulation, with S_8 fairing much worse than S_2 .

This system can end in several possible ways. The end result is not immediately apparent and it is dependent on the random folding mechanism. Systems with very large populations are less affected by changes than smaller systems are, and this allows more equal distribution of string selection and folding selection. The relatively equal selection process makes the system continue along its current trajectory (not shown). The lowest two subsets, (S_{10}, S_{12}) and (S_2, S_4, S_8) , continue on their downward path until their demise. This can take several thousand more generations, as thriving strings sometimes interact to produce weaker strings, keeping lesser subsets from slipping away. During this time (S_{15}) , (S_3, S_5) and (S_1) concentrations increase, and the weaker subset of (S_{10}, S_{12}) fall away. In this particular system (S_3, S_5) and (S_1) find themselves unable to cope with the resulting exploiter concentration, and S_1 is the first to be reduced to zero, followed by (S_3, S_5) , leaving S_{15} as the only dominant string. However, this is not

4.3. Random and Selective Homogeneous Systems

the only possible outcome of the system, and the resulting organization of strings, $[(S_{15})(S_3, S_5)(S_1)]$, can exist, and end, in multiple states. The state of the organization is affected by the presence of strings not from the organization, the size of the population, and the concentrations of organization members that result from initial reactions of the system. This organization was further investigated in systems initialized with the subgroup and different populations sizes (10^3 , 10^4 , and 10^5 strings)(not shown). The organization has at least three states, and it is capable of switching between states autonomously. The most stable state often occurs in larger systems (10^6). It is characterised by all four members fluctuating around 25%, with relatively equal string concentrations of S_3 and S_5 , which act as buffers between S_1 and S_{15} . S_1 and S_{15} fluctuate much more than S_3 and S_5 , and S_1 and S_{15} concentrations vary between 15% and 35%. The two other states are variations of the first state, they are inverse states of each other, and they are more likely to occur in systems with a population of less than 10^6 strings. They are characterised by a dominance of either S_1 or S_{15} for a prolonged period, and the concentrations of S_3 and S_5 are further apart from each other while together they average less than 25% of the total population. In smaller systems, stable states are capable of spontaneously changing into one of the states partially dominated by S_1 or S_{15} . If the system becomes partially dominated by S_1 or S_{15} it can either regress back to the stable state, or it can progress further into a single string system dominated by either S_1 or S_{15} .

0110 selective folding

The S_6 [0110] selective folding system initially develops in a similar way to the S_6 random folding system. Selective folding is explained in detail in

4.3. Random and Selective Homogeneous Systems

Chapter 3 under Sequence Folding: Mapping. Several subgroups initially diversify but the system takes a new trajectory after a few generations. The initial string population is partially metabolised into S_9 , leading to the production of nine other strings: $S_1, S_2, S_3, S_4, S_5, S_8, S_{10}, S_{12}$ and S_{15} . Figs. 4.10.a and 4.10.b show two time frames from the same simulation, each initialised with S_6 and with a population of 10^5 strings. The system was only allowed to utilize selective folding.

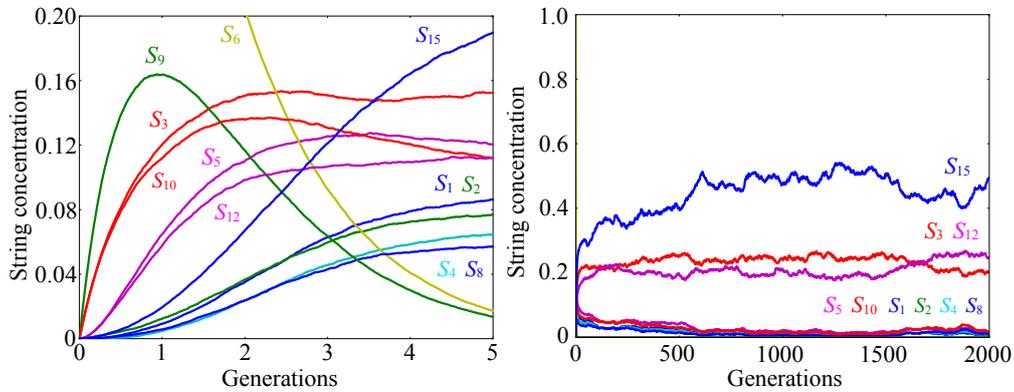


Figure 4.10: (a) left, S_6 - F_{sel} , $M = 10^5$, 5 generations. (b) right, S_6 - F_{sel} , $M = 10^5$, 2000 generations.

The initial reactions of the S_6 selective folding system are very similar to the reactions that take place in the S_6 random folding system. They lead to the production of S_9, S_3 , and S_{10} , with S_9 initially produced at twice the rate of S_3 and S_{10} . Selective folding initially selects each folding type in an equal ratio, but when an operator produces itself (active replication) it becomes more likely to use the folding type that allowed active replication. There are no active replications in the initial S_6 reactions, therefore S_6 does not become partial to any folding type when acting on itself.



4.3. Random and Selective Homogeneous Systems

The initial reactions of the S_6 selective system are exactly the same as in the S_6 random folding system (Fig. 4.9). The tables below shows the strings produced when S_6 interacts with itself and the initial strings it produces: S_9 , S_3 , and S_{10} . Their interactions still lead to the introduction of S_2 , S_5 , S_{12} , and S_{15} .

	$\oplus_{6\cdot cf}$	$\oplus_{6\cdot tc}$	$\oplus_{6\cdot tp}$	$\oplus_{6\cdot tt}$
S_6	S_9	S_9	S_3	S_{10}
S_9	S_6	S_6	S_{12}	S_{10}
S_3	S_3	S_3	S_3	S_2
S_{10}	S_5	S_5	S_{15}	S_{10}

When S_6 operates canonically and transposed canonically on S_9 it produces itself. The active replication causes the string memory for S_6 to increase for that folding type, which increases the chance of that folding type being selected for S_6 in the future. S_3 is produced often via passive replication when S_6 operates on it, but this does not increase the chance of it being selected.

	$S_{6\oplus cf}$	$S_{6\oplus tc}$	$S_{6\oplus tp}$	$S_{6\oplus tt}$
\oplus_6	S_9	S_9	S_3	S_{10}
\oplus_9	S_6	S_6	S_{12}	S_5
\oplus_3	S_5	S_{12}	S_5	S_{12}
\oplus_{10}	S_3	S_{10}	S_9	S_9

When the initial strings produced act on S_6 , S_{10} is the only operator that displays active replication. When S_{10} operates on S_6 transposed canonically is actively replicates, producing S_{10} , and increasing S_{10} 's future chances to fold transposed canonically. Other active replications that are able to take place but are not shown include:

$$S_6 \oplus_{cf,tc} S_9 = S_6$$

4.3. Random and Selective Homogeneous Systems

$$\begin{aligned}
 S_9 \oplus_{cf,tc} S_9 &= S_9 \\
 S_3 \oplus_{tc,tt} S_3, S_9 &= S_3 \\
 S_{10} \oplus_{tc} S_3, S_9, S_{10} &= S_{10}
 \end{aligned}$$

It is immediately clear that some strings benefit more from certain folding types, and others have a distributed preference. S_6 and S_9 are partial towards both canonical and transposed canonical folding types, with one reaction promoting each folding type. S_3 has four reactions promoting two folding types, S_3 operates on both itself and S_9 , transposed canonically and transposed topologically, to produce itself. S_{10} operates transposed canonically on S_3 , S_9 , and itself, promoting one folding type.

	$\oplus_{6\cdot cf}$	$\oplus_{6\cdot tc}$	$\oplus_{6\cdot tp}$	$\oplus_{6\cdot tt}$
S_2	S_1	S_1	S_3	S_2
S_5	S_{10}	S_{10}	S_0	S_{10}
S_{12}	S_{12}	S_{12}	S_{12}	S_8
S_{15}	S_{15}	S_{15}	S_{15}	S_{10}

	$S_{6\oplus cf}$	$S_{6\oplus tc}$	$S_{6\oplus tp}$	$S_{6\oplus tt}$
\oplus_2	S_1	S_8	S_1	S_8
\oplus_5	S_{12}	S_5	S_6	S_6
\oplus_{12}	S_{10}	S_3	S_{10}	S_3
\oplus_{15}	S_{15}	S_{15}	S_{15}	S_{15}

None of the reactions with the next set of strings produced were beneficial for S_6 as an operator. Although S_{12} passively replicates when S_6 operates on it, passive replication is not a criterium for improving folding selection. As for the second set of strings acting on S_6 , only S_5 and S_{15} actively replicate. S_5 operates transposed canonically on S_6 to produce itself. S_{15} produces itself from S_6 with all folding types. This equally increases the selection chance for each folding type, which maintains an equal chance

4.3. Random and Selective Homogeneous Systems

of selection for each folding type, and simultaneously makes S_{15} benefit less from more specific reactions with other strings. More active replications become possible with the introduction of new strings.

$$\begin{aligned}
 S_2 \oplus_{tc,tt} S_3, S_9 &= S_2 \\
 S_5 \oplus_{tp,tt,tc} S_5 &= S_5 \\
 S_5 \oplus_{tc} S_6, S_9, S_{10}, S_{15} &= S_5 \\
 S_{10} \oplus_{tp,tt,tc} S_5 &= S_{10} \\
 S_{10} \oplus_{tc} S_6, S_9, S_{10}, S_{15} &= S_{10} \\
 S_{12} \oplus_{tc,tt} S_9, S_{12} &= S_{12} \\
 S_{15} \oplus_{cf,tc,tp,tt} S_5, S_6, S_9, S_{10}, S_{15} &= S_{15}
 \end{aligned}$$

S_2 selects for two folding types when operating on S_3 and S_9 . S_5 selects for three folding types when it operates on itself, and it is very partial towards the transposed canonical folding type, using it to actively replicate with four other strings. Operator S_{10} behaves exactly the same as S_5 , operating on the exactly the same strings, with the same respective folding types. S_{12} actively replicates uses two folding types, each applied to two different strings. S_{15} , true to its exploitive nature, utilizes all four folding types equally over a group of five strings to produce itself.

Fig. 4.10.a is almost exactly the same as Fig. 4.9.a. After one generation the S_9 population has already peaked and is declining. S_3 and S_{10} increase next until they peak between generations two and three, while S_5 and S_{12} increase and peak between generations three and four. The following two strings are S_{15} and S_2 , and they are followed by S_1 , S_8 and S_4 .

	\oplus_{6-cf}	\oplus_{6-tc}	\oplus_{6-tp}	\oplus_{6-tt}
S_1	S_2	S_2	S_0	S_2
S_8	S_4	S_4	S_{12}	S_8
S_4	S_8	S_8	S_0	S_8

4.3. Random and Selective Homogeneous Systems

	$S_{6\oplus cf}$	$S_{6\oplus tc}$	$S_{6\oplus tp}$	$S_{6\oplus tt}$
\oplus_1	S_4	S_4	S_4	S_4
\oplus_8	S_2	S_2	S_8	S_1
\oplus_4	S_8	S_1	S_2	S_2

None of the reactions involving S_6 operating on S_1 , S_8 and S_4 are active replications. Of the strings that operate on S_6 , only S_8 actively replicates when operating topologically on S_6 . The addition of S_1 , S_8 and S_4 to the population increases the number of possible active replications.

Selective folding systems have access to the same folding types as random folding systems, because of this the subset of strings that S_6 produces in the S_6 selective system is the same as in the S_6 random system. As such, none of the reactions between the progeny of S_6 add any new strings to the population. The only outstanding strings are once again the destructor, and the structurally related subset of strings containing S_7 , S_{11} , S_{13} and S_{14} . Below are all possible active replications that can occur when all strings in this subset are present in the soup together.

$$S_1 \oplus_{cf,tc,tp,tt} S_1, S_3, S_9 = S_1$$

$$S_2 \oplus_{tc,tt} S_1, S_3, S_9 = S_2$$

$$S_3 \oplus_{tc,tt} S_1, S_3, S_9 = S_3$$

$$S_4 \oplus_{tc} S_8, S_9, S_{12} = S_4$$

$$S_5 \oplus_{tc,tp,tt} S_5 = S_5$$

$$S_5 \oplus_{tc} S_6, S_9, S_{10}, S_{15} = S_5$$

$$S_6 \oplus_{cf,tc} S_9 = S_6$$

$$S_8 \oplus_{cf,tc} S_8, S_9, S_{12} = S_8$$

$$S_8 \oplus_{tp} S_4, S_6, S_{12} = S_8$$

$$S_9 \oplus_{cf,tc} S_9 = S_9$$

$$S_{10} \oplus_{tc,tp,tt} S_5 = S_{10}$$

$$S_{10} \oplus_{tc} S_6, S_9, S_{10}, S_{15} = S_{10}$$

 4.3. Random and Selective Homogeneous Systems

$$S_{12} \oplus_{tc,tt} S_8, S_9, S_{12} = S_{12}$$

$$S_{15} \oplus_{cf,tc,tp,tt} S_5, S_6, S_9, S_{10}, S_{15} = S_{15}$$

It is clear from the reactions above that actively replicating operators use specific groups of strings that have the structural elements they require for active replication. S_1 benefits equally from all folding types when operating on S_1 , S_3 , and S_9 . S_2 and S_3 also benefit from operating on S_1 , S_3 , and S_9 , but only when operating transposed canonically or transposed topologically. S_4 benefits from S_8 , S_9 , and S_{12} , but only when operating transposed canonically. The reactions for S_5 do not change, and it benefits when operating on itself transposed canonically, topologically, and transposed topologically, and it can actively replicate with S_6 , S_9 , S_{10} , and S_{15} when folded transposed canonically. There are no new reactions for S_6 , and the only active replications it can perform involve S_9 . S_8 actively replicates using itself, S_9 , and S_{12} when operating canonically and transposed canonically, and it utilizes S_4 , S_6 , and S_{12} when operating topologically. The reactions for S_9 remain the same, and the only active replications that it can perform are self-reactions, which utilize canonical and transposed canonical folding. S_{10} does also not gain any new self-replications, and it still uses exactly the same strings as S_5 in the same folding conformations. S_{12} gains the ability to replicate using S_8 in addition to itself and S_9 when folded transposed canonically or transposed topologically. S_{15} does not gain any new self-replications and remains able to replicate using all folding types when operating on itself, S_5 , S_6 , S_9 , and S_{10} .

The initial growth patterns in Figs. 4.10.a and 4.9.a are very similar. The S_9 population grows rapidly, but there are not enough reactions to maintain its concentration, and the S_9 population begins declining before the end of the first generation. In Fig. 4.9.a, the next two emergent string

 4.3. Random and Selective Homogeneous Systems

pairs, S_3/S_{10} and S_5/S_{12} , each split and began re-merging towards the end of the fifth generation, with S_3/S_5 eventually dominating S_{10}/S_{12} . However, in Fig. 4.10.a, S_5 begins declining and does not remain with S_3 . Instead, S_5 decreases until it reaches the concentration of S_{10} and S_{12} , and S_{12} increases until it reaches the concentration of S_3 . In Fig. 4.10.b, as opposed to Fig. 4.9.b, the strings with structural similarity support, or at least join, each other in the same concentration ranges. The S_{15} concentration has also increased from 16% in Fig. 4.9.a to almost 20% in Fig. 4.10.a. The group of structurally related strings, S_1 , S_2 , S_4 , and S_8 , all begin declining between the fifth and sixth generations. S_1 remains above the others, while S_2 pairs with S_4 and S_8 remains the string with the lowest concentration.

In Fig. 4.10.b the most dominant groups over a long period are S_{15} and (S_3, S_{12}) . S_3 and S_{12} are structurally similar, each containing two one bits that are adjacent to each other and occupy one half of the string. Even though S_3 and S_{12} are related, it is strange that they are dominating together, as they do not support each other much in random systems, only depending 7.1% on each other. It is also interesting that over 2000 generations there are two very weak subsets that persist, (S_5, S_{10}) and (S_1, S_2, S_4, S_8) . Why do two strings that don't support each other thrive so well, and why do two struggling subsets seem to persist indefinitely? S_{15} is the answer to both of these questions. S_{15} has no folding preference in this system, producing itself from the same set of strings, regardless of folding type. This gives S_{15} no advantage or disadvantage, and selection and folding preference essentially have no effect on it. S_{15} has an early advantage by being able to replicate using S_6 and S_9 , but by the third generation S_6 and S_9 have depleted and they average less than 10%. At the same time two other strings that S_{15} can use to replicate, S_5 and S_{10} , have increased

4.3. Random and Selective Homogeneous Systems

to average above 10%. The constant availability of replication partners and the self-replication rate of S_{15} allow it to continue increasing until it reaches a state where it remains between 40% and 50% of the population.

S_{15} is also key to the success of S_3 and S_{12} . S_3 and S_{12} are passive replicators when operated on by S_{15} , and they are also each 21.4% dependent on S_{15} . This places S_3 and S_{12} in direct competition with each other for being operated on by S_{15} : the string that is operated on the most during the simulation will have the highest concentration. When S_3 and S_{12} operate on S_{15} with a transposed folding type they both produce S_{15} , and when they operate using non-transposed folding types they respectively produce S_5 and S_{10} . Out of all reactions involving S_{15} and, S_3 or S_{12} , half respectively produce S_3 or S_{12} , one quarter produce S_{15} and one quarter respectively produce S_5 or S_{10} .

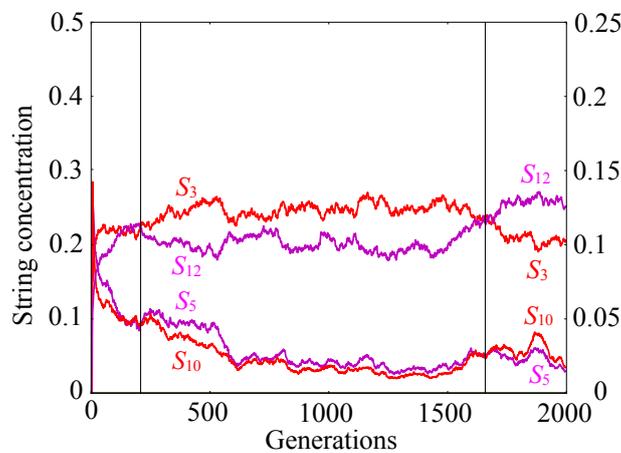


Figure 4.11: S_6 - F_{ran} , $M = 10^5$, 2000 generations. Only S_3 , S_5 , S_{10} and S_{12} are shown. S_3 and S_{12} concentrations are on the left, S_5 and S_{10} and concentrations are on the right.

Fig. 4.11 is a magnified version of Fig. 4.10.b, with only S_3 , S_5 , S_{10} and S_{12} displayed. The two subsets are visualised on different scales, with S_3

 4.3. Random and Selective Homogeneous Systems

and S_{12} concentrations displayed on the left, while S_5 and S_{10} concentrations are scaled the right. The effect of S_3 and S_{12} concentrations visibly effect S_5 and S_{10} respectively, with an increased S_3 promoting S_5 and an increased S_{12} promoting S_{10} . However, S_3 and S_{12} are both partial towards transposed folding types, as their self-reactions are self-replications. These folding types also promote S_{15} , and not S_5 or S_{10} . When S_3 and S_{12} self-replicate canonically or topologically they respectively produce S_1 and S_8 , which significantly contributes to the persistent survival of the subgroup (S_1, S_2, S_4 , and S_8).

As S_3 and S_{12} adapt, over time, towards transposed folding types, they become increasingly likely to produce themselves and S_{15} , and less likely to produce S_5 and S_{10} , or S_1 and S_8 . After thousands of generations subgroups will fade completely, and only occasionally will they be produced, only to be extinguished again. The system eventually reaches a state where S_3 and S_{12} operate almost exclusively using transposed folding types, while S_{15} has retained no preferred folding type.

	S_3	S_{12}	S_{15}
$\oplus_{3-(tc,tt)}$	S_3	S_{12}	S_{15}
$\oplus_{12-(tc,tt)}$	S_3	S_{12}	S_{15}
$\oplus_{15-(cf,tc,tp,tt)}$	S_3	S_{12}	S_{15}

The end state in this system is similar to the end state of the S_6 topological folding system, where strings that maintain the end population of the system are composed of passive self-replicators. The only differences between the end states are: the survival of different string species, and all strings that remain in the S_6 selective folding system have evolved to be only passive- and self-replicators. The end state of this system is extremely stochastic, and the survival of the strings is up to chance, but the surviving

4.3. Random and Selective Homogeneous Systems

system is also robust, and depletion of any of the strings will not affect the relationship of the remaining strings.

1001 random folding

The S_9 [1001] random folding string system is an example of a random system that becomes stable and remains that way for a potentially infinite amount of time. After an initial phase a closed subgroup of strings has emerged. The initial string population is metabolised into S_3 and S_5 , which leads to the production of S_1 and S_{15} . Figs. 4.12.a and 4.12.b show two time frames from the same simulation, initialised with S_9 , and with a population of 10^5 strings. The system was only allowed to utilize random folding.

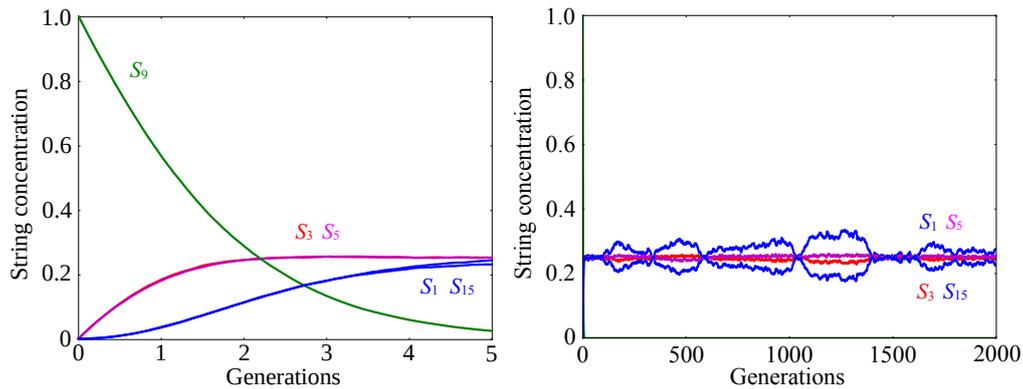


Figure 4.12: (a) left, S_9 - F_{ran} , $M = 10^5$, 5 generations. (b) right, S_9 - F_{ran} , $M = 10^5$, 2000 generations.

The initial self-reactions of S_9 lead to the production of itself, S_3 , and S_5 . As in the S_6 random folding system, S_9 is initially produced at twice the rate of two other strings, S_3 and S_5 . This is because there are two S_9 self reactions that produce S_9 , and S_3 and S_5 are only produced by one self-reaction each. Random folding systems will select each folding type relatively equally over long simulations.

4.3. Random and Selective Homogeneous Systems

	$\oplus_{9.cf}$	$\oplus_{9.tc}$	$\oplus_{9.tp}$	$\oplus_{9.tt}$
S_9	S_9	S_9	S_3	S_5

The canonical and transposed canonical self-reactions both produce S_9 , while the topological operation produces S_3 and the transposed topological operation produces S_5 . There is a high probability that the two newly produced strings will encounter S_9 in the string population before they encounter each other, and there is an equal chance that they will encounter S_9 as either an operator or as a string.

	$\oplus_{9.cf}$	$\oplus_{9.tc}$	$\oplus_{9.tp}$	$\oplus_{9.tt}$
S_3	S_3	S_3	S_3	S_1
S_5	S_5	S_5	S_{15}	S_5
S_9	S_9	S_9	S_3	S_5

The table above shows the strings produced when S_3 , S_5 , and S_9 are operated on by S_9 for each folding method. Their interactions lead to the introduction of S_1 and S_{15} . The following table contains the strings produced when S_9 is operated on by S_3 and S_5 for each folding type.

	$S_{9\oplus cf}$	$S_{9\oplus tc}$	$S_{9\oplus tp}$	$S_{9\oplus tt}$
\oplus_3	S_5	S_3	S_5	S_3
\oplus_5	S_3	S_5	S_9	S_9

No new strings were produced when S_3 and S_5 operated on S_9 . All operations by S_3 on S_9 produced S_3 and S_5 , and S_5 operating on S_9 produced S_3 , S_5 , and S_9 . New strings were only introduced into the system by S_9 operating on S_3 and S_5 . The following two tables contain the reactions of S_9 operating on S_1 and S_{15} , and S_9 being operated on by S_1 and S_{15} .

	$\oplus_{9.cf}$	$\oplus_{9.tc}$	$\oplus_{9.tp}$	$\oplus_{9.tt}$
S_1	S_1	S_1	S_3	S_1
S_{15}	S_{15}	S_{15}	S_{15}	S_5

4.3. Random and Selective Homogeneous Systems

	$S_{9\oplus cf}$	$S_{9\oplus tc}$	$S_{9\oplus tp}$	$S_{9\oplus tt}$
\oplus_1	S_1	S_1	S_1	S_1
\oplus_{15}	S_{15}	S_{15}	S_{15}	S_{15}

None of the reactions lead to the introduction of new strings, and only two of the eight reactions were not replication reactions. Both S_1 and S_{15} entirely actively replicate off of S_9 , and they are both successful passive replicators as well.

Another similarity between the S_6 and S_9 random folding systems is that all reactions involving the initial string, as either an operator or a string, produced all strings in the systems, and reactions between produced strings did not add any new strings to the system. A large difference between the systems is the amount of subsets produced by the initial string. The S_6 system produces three subsets, and the S_9 system only produces one subset. The subset produced in the S_9 random system is also the dominant subset in the S_6 random system, and because it is the only subgroup produced in the S_9 system it is able to develop into a semi-stable state without interference from other strings or subgroups. The S_9 string itself does not survive, because it does not have enough reactions producing it, and all reactions that produce it are passive or active replications, meaning that no other strings produce S_9 when interacting. S_9 decays noticeably slower than other initial strings, but it eventually fades from the system before ten generations, leaving only the four strings it produced.

Fig. 4.12.b shows the behaviour of the system over a 2000 generations: periods of almost equal string concentrations, frequently interspersed by partial dominance of S_1 or S_{15} . The dependence tables for this state of the S_9 system revealed that S_1 and S_{15} are not dependent on each other at all, and that S_1 and S_{15} are each 53.1% dependant on self-replication. They

4.3. Random and Selective Homogeneous Systems

also inversely rely on S_3 and S_5 , with S_1 relying 34.4% on S_3 and 12.5% on S_5 , and S_{15} depending 34.4% S_5 and 12.5% on S_3 .

S_3 and S_5 also depend equally on themselves and on each other, and inversely on S_1 and S_{15} . They both depend 40.6% on themselves and 12.5% on each other, while S_3 depends 25% on S_{15} and S_5 depends 25% on S_1 , and S_3 depends 21.9% on S_1 and S_5 depends 21.9% on S_{15} . S_{15} produces S_3 eight times and S_5 seven times, and S_1 produces S_3 seven times and S_5 eight times. It is this slight difference in dependence of S_3 and S_5 on S_1 and S_{15} that causes S_3 to be favoured when S_{15} is dominating, and S_5 to be favoured when S_1 is dominating.

This system is also interesting because in this subgroup all members share an equal portion of control, where control is the total amount that string is depended on. If S_1 depends 34.4% on S_3 and 53.1% itself, then S_3 has 34.4% control over S_1 , and S_1 has 53.1% control over itself. In Fig. 4.9.b the control is not equally distributed between strings, and some strings, such as S_{15} had greater than 100% total control, and others had less than 100% total control. While a string will and must always have a total dependence of 100%, a string can have more or less than 100% control. This simply means some strings may have more influence than other strings, even if they have less than 100% total control. Control is also often concentrated in specific subgroups. However, in Fig. 4.12.b the control is equally distributed between the four strings, and each string has 100% control. The equal distribution of control, combined with a large reactor vessel, allowed this system of strings to continue indefinitely without a single string dominating. In smaller populations of 10^3 strings, S_1 or S_{15} will usually dominate before 10^3 generations. Smaller populations lower the amount of strings needed to reach the threshold for domination, which is

4.3. Random and Selective Homogeneous Systems

usually 50%. However, in larger populations, the threshold is never reached, because random selection ensures other strings will be selected and the dominant string will continuously be displaced away from the domination threshold.

1001 selective folding

The S_9 [1001] selective folding system initially develops in exactly the same way as the S_9 random folding system. The same subgroup of four strings as in 4.12 appear, but the group divides into two subgroups that compete against each other. The initial string population is metabolised from S_9 into S_1 , S_3 , S_5 and S_{15} . Figs. 4.10.a and 4.10.b show two time frames from the same simulation each initialised with S_9 and with a population of 10^5 strings. The system was only allowed to utilize selective folding.

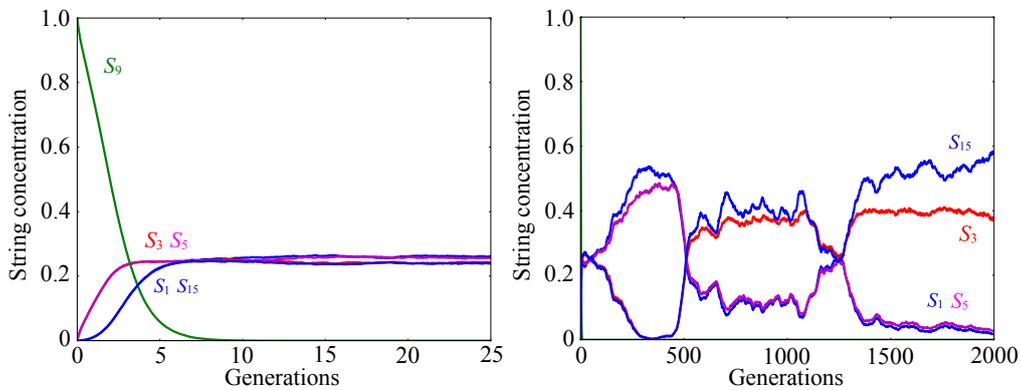


Figure 4.13: (a) left, S_9 - F_{sel} , $M = 10^5$, 25 generations. (b) right, S_9 - F_{sel} , $M = 10^5$, 2000 generations.

The initial reactions of the S_9 selective folding system are also identical to the reactions that take place in the S_9 random folding system, and they lead to the production of S_1 , S_3 , S_5 , and S_{15} . S_9 is initially produced at twice the rate of S_3 and S_5 , but this does not stop the decline and eventual

4.3. Random and Selective Homogeneous Systems

demise of S_9 . Selective folding initially selects each folding type in an equal ratio, but folding types that favour active replication are selected for, and those folding types gain an increased chance for being selected. There are two active replications in the initial S_9 reactions, and S_9 becomes partial to the canonical and transposed canonical folding types during self reactions.

	$\oplus_{9.cf}$	$\oplus_{9.tc}$	$\oplus_{9.tp}$	$\oplus_{9.tt}$
S_9	S_9	S_9	S_3	S_5

The strings produced in the S_9 selective system are exactly the same the S_9 random folding system (Fig. 4.12); the following tables show the strings produced when S_9 interacts with itself and the initial strings it produces: S_1 , S_3 , S_5 , and S_{15} . The interactions between the produced strings do not lead to the production of any new strings.

	$\oplus_{9.cf}$	$\oplus_{9.tc}$	$\oplus_{9.tp}$	$\oplus_{9.tt}$
S_1	S_1	S_1	S_3	S_1
S_3	S_3	S_3	S_3	S_1
S_5	S_5	S_5	S_{15}	S_5
S_9	S_9	S_9	S_3	S_5
S_{15}	S_{15}	S_{15}	S_{15}	S_5

	$S_{9\oplus cf}$	$S_{9\oplus tc}$	$S_{9\oplus tp}$	$S_{9\oplus tt}$
\oplus_1	S_1	S_1	S_1	S_1
\oplus_3	S_5	S_3	S_5	S_3
\oplus_5	S_3	S_5	S_9	S_9
\oplus_9	S_9	S_9	S_3	S_5
\oplus_{15}	S_{15}	S_{15}	S_{15}	S_{15}

Out of all reactions with S_9 , there are only four reactions that produce S_9 , and the only active replications are the two S_9 self-replications. Even though active replication causes the string memory for S_9 to increase for

4.3. Random and Selective Homogeneous Systems

the canonical and transposed canonical folding types, it is not enough to permanently sustain the S_9 population.

There are passive and active replications among the initial reactions between S_9 and the strings it produces, and strings operating on S_9 all display some degree of active replication. When S_1 and S_{15} operate on S_9 they actively replicate with all folding types, leading to no one type being preferred over any other. S_3 prefers both transposed folding types when operating on S_9 , and S_5 only selects for transposed canonical folding when operating on S_9 . The following table contains all active replications that are able to take place during the initial phase, including those that are not shown:

$$\begin{aligned}
 S_1 \oplus_{cf,tc,tp,tt} S_1, S_3, S_9 &= S_1 \\
 S_3 \oplus_{tc,tt} S_1, S_3, S_9 &= S_3 \\
 S_5 \oplus_{tc,tp,tt} S_5 &= S_5 \\
 S_5 \oplus_{tc} S_9, S_{15} &= S_5 \\
 S_9 \oplus_{cf,tc} S_9 &= S_9 \\
 S_{15} \oplus_{cf,tc,tp,tt} S_5, S_9, S_{15} &= S_{15}
 \end{aligned}$$

From the data above it is clear that S_1 and S_{15} do not have any preference for folding types, but the other three strings are more selective in their folding preference. All strings benefit from S_9 as if it were another string they replicate off of. This implies that S_9 does not influence the folding preference of the strings that operate on it any differently from other strings they operate on, i.e., S_9 affects the folding preferences of S_1 and S_3 similarly to S_1 and S_3 . As a result, when S_9 is depleted, the folding preferences of the other strings do not change because of the disappearance of S_9 . The following table contains the active replications that occur in the final state of the system, in the absence of S_9 , which occurs around the tenth generation:

4.3. Random and Selective Homogeneous Systems

$$S_1 \oplus_{cf,tc,tp,tt} S_1, S_3 = S_1$$

$$S_3 \oplus_{tc,tt} S_1, S_3 = S_3$$

$$S_5 \oplus_{tc,tp,tt} S_5 = S_5$$

$$S_5 \oplus_{tc} S_{15} = S_5$$

$$S_{15} \oplus_{cf,tc,tp,tt} S_5, S_{15} = S_{15}$$

In Fig. 4.13.a, S_9 operators select for the two folding types that promote S_9 active replication, canonical and transposed canonical folding. These two folding types will also make S_9 promote whichever string it operates on, making S_9 promote passive replication of the other strings. However, S_9 cannot sustain itself, and the S_9 population depletes around the tenth generation.

S_1 is partial towards all folding types when it operates on itself and on S_3 , and it will always produce itself when it operates on itself or S_3 . S_1 will also always produce S_5 when it operates on S_5 or on S_{15} .

S_3 actively replicates when operating on the initial string, S_9 , using transposed canonical or transposed topological folding types. This leads to S_3 selecting for these folding types from the beginning of the simulation, which it will benefit from later, because it also uses those two folding types to actively replicate when operating on itself and on S_1 . When S_3 operates with one of its two preferred folding types on S_5 or S_{15} it will always produce S_{15} .

S_5 self-replicates with all folding types except canonical folding, and three of the five S_5 active-replications are self-replications. The other two active replications occur when S_5 operates transposed canonically on S_9 or S_{15} . S_5 will initially select for the transposed canonical folding type because of the abundance of S_9 , and this will benefit S_5 when S_{15} becomes

4.3. Random and Selective Homogeneous Systems

abundant. When S_5 operates transposed canonically on S_1 or S_3 it will produce S_1 , and S_5 promotes passive replication among all strings when it operates topologically or transposed topologically.

S_{15} is partial towards all folding types when it operates on itself, S_5 , and S_9 . When S_{15} operates on S_1 or S_3 it will always produce S_3 , regardless of the folding type. The following table contains the active replications that take place in Fig. 4.13.b.

	S_1	S_3	S_5	S_{15}
$\oplus_{1-(cf,tc,tp,tt)}$	S_1	S_1	S_5	S_5
$\oplus_{3-(tc,tt)}$	S_3	S_3	S_{15}	S_{15}
$\oplus_{5-(tc)}$	S_1	S_1	S_5	S_5
$\oplus_{5-(tp,tt)}$	S_1	S_3	S_5	S_{15}
$\oplus_{15-(cf,tc,tp,tt)}$	S_3	S_3	S_{15}	S_{15}

The above table illustrates how the population becomes divided into two groups. The first group consists of S_1 and S_5 , and the second group consists of S_3 and S_{15} . Both groups have similar properties, and members display passive replication when operating on strings of the same group, and active replication or partner beneficial reactions when operating on members of the other group. Each group is also able to exist independently from the other group as a semi-stable passive self-replicator network.

$\oplus_{1-(cf,tc,tp,tt)}$	S_1	S_3	S_5	S_{15}	\rightarrow	$\oplus_{1-(cf,tc,tp,tt)}$	S_1	S_5
$\oplus_{5-(tc)}$	S_1	S_1	S_5	S_5		$\oplus_{5-(tc)}$	S_1	S_5
$\oplus_{3-(tc,tt)}$	S_3	S_3	S_{15}	S_{15}	\rightarrow	$\oplus_{3-(tc,tt)}$	S_3	S_{15}
$\oplus_{15-(cf,tc,tp,tt)}$	S_3	S_3	S_{15}	S_{15}		$\oplus_{15-(cf,tc,tp,tt)}$	S_3	S_{15}

Fig. 4.13.b goes through three distinct phases, each dominated by a string group. The first phase is dominated by S_1 and S_5 , and the second and third phases are dominated by S_3 and S_{15} . The two groups are unable to coexist, because members of one group are able to actively replicate off of members from the other, placing the groups in direct competition.

This system usually ends with one string eventually dominating the population, but these systems can also continue for thousands of generations without reaching an end state. The two string groups will constantly try and suppress each other and until one group has been eliminated. When this occurs the system enters a completely stochastic state where the successor of the the remaining strings will be determined by random selection process, leaving the fate of the system to random chance.

4.4 Heterogeneous Systems

In the following section heterogeneous systems were divided into four categories, based on how the destructor and exploiter string were handled. Within each category every run was performed once for each folding type, including random and selective folding methods.

Destructor [0000] production prohibited

Each of the six systems in the following section were initialized with a random string population, which was generated by randomly selecting strings from an available list and repeating until the desired population count is reached. The only filter applied in this section is the inhibition of destructor creation. All reactions that would produce destructors are considered not to take place and their iteration ‘turns’ are restarted. In each of the

4.4. Heterogeneous Systems

systems, except F_{sel} , diversification of subgroups occurs relatively quickly, and within five to ten generations each subgroup settles on an initial concentration. Figs. 4.14–4.16 show six simulations each initialised with a heterogeneous population of 10^5 strings. Each system utilized a different folding mechanism.

Figs. 4.14.a and 4.14.b show two systems, each initialised with all strings. Fig. 4.14.a used canonical folding and Fig. 4.14.b used transposed canonical folding. Both systems displayed exactly the same behaviour: the exploiter string dominates as a lone string in the population, and on average it will account for 23% of the population. Two subgroups vie for control of the remaining space in the system. Two-bit subgroup members, S_3 , S_5 , S_{10} , and S_{12} , occupy on average 12.5% each, and make up 50% of the entire population. One-bit subgroup members, S_1 , S_2 , S_4 , and S_8 , occupy on average 6.8% each, and combined they account for 27% of the entire population.

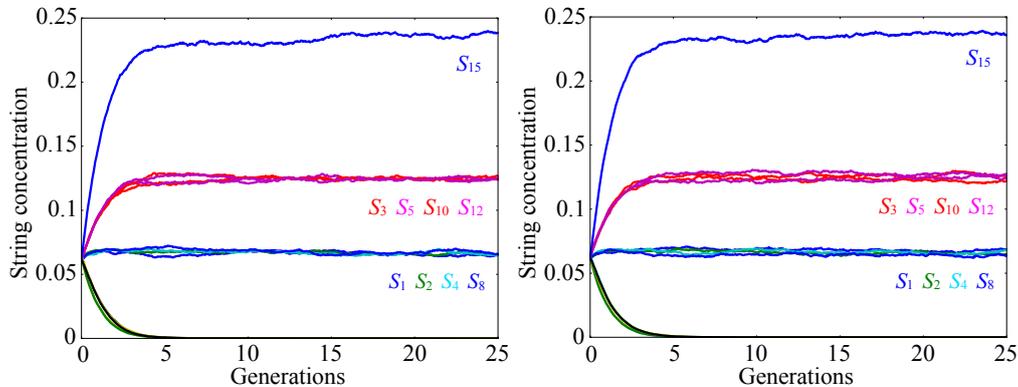


Figure 4.14: Only the destructor was not allowed to replicate in this system. (a) left, S_X-F_{can} , $M = 10^5$, 25 generations. (b) right, $S_X-F_{t.can}$, $M = 10^5$, 25 generations.

Fig. 4.15.a used topological folding and Fig. 4.15.b used transposed topological folding. These two systems displayed different behaviour from each other and from the previous two systems. In Fig. 4.15.a the subgroup of S_5

4.4. Heterogeneous Systems

and S_{10} dominate the population and average 18.3% each. The exploiter is next most prolific string at 11.4%. It is closely followed by the one-bit subgroup, S_1 , S_2 , S_4 , and S_8 , averaging 9.9% each. The last surviving subgroup in Fig. 4.15.a is the remainder of the two-bit subgroup, consisting of S_3 and S_{12} , which average 6.2% each.

Only three strings are able to thrive in Fig. 4.15.b, the exploiter and half of the two-bit subgroup containing S_3 and S_{12} . The exploiter initially plateaus at 46.4%, while S_3 and S_{12} average 26.8% each. This system reaches a stochastic end state that contains passive self-replicators, the same end state as the S_6 selective folding system, but with transposed topological folding.

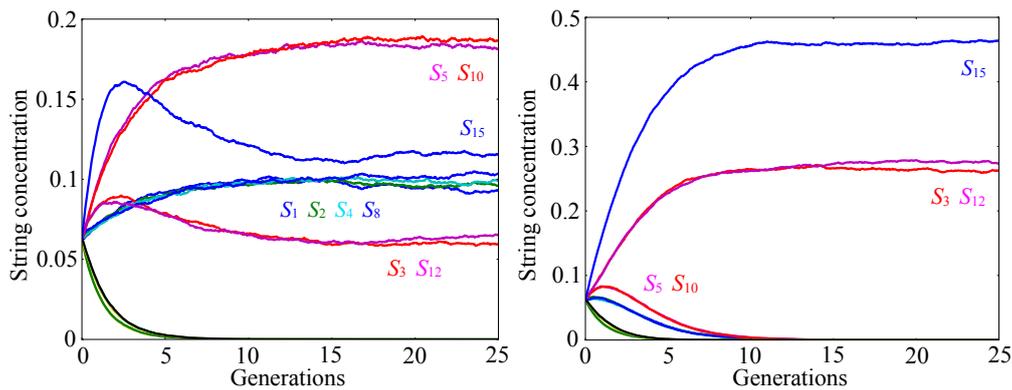


Figure 4.15: Only the destructor was not allowed to replicate in this system. (a) left, S_X-F_{top} , $M = 10^5$, 25 generations. (b) right, $S_X-F_{t.top}$, $M = 10^5$, 25 generations.

Fig. 4.16.a used random folding and Fig. 4.16.b used selective folding. Fig. 4.16.a displays exactly the same initial behaviour as Fig. 4.14.a: the exploiter string dominates the population as the string with the single highest concentration, reaching 21.6% by the 25th generation. It is followed by two-bit subgroup, S_3 , S_5 , S_{10} , and S_{12} , averaging 12.4% each. Last is the

one-bit subgroup, containing S_1 , S_2 , S_4 , and S_8 , which average 7.2% each by the 25th generation.

Fig. 4.16.b is similar to Fig. 4.15.b, where the exploiter and two substrings, S_3 and S_{12} , dominate the population. However, in Fig. 4.16.b, there are two other groups that survive for an extended period as well, S_5 and S_{10} , and one-bit subgroup S_1 , S_2 , S_4 and S_8 . By 100 generations the exploiter string has reached 44.2% , and S_3 and S_{12} average 21.3% each. After 100 generations the group containing the remaining two-bit strings, S_5 and S_{10} , has reached 3.3%. The last group, containing the one-bit strings, has managed to continue existing after 100 generations, albeit at only 1.6%.

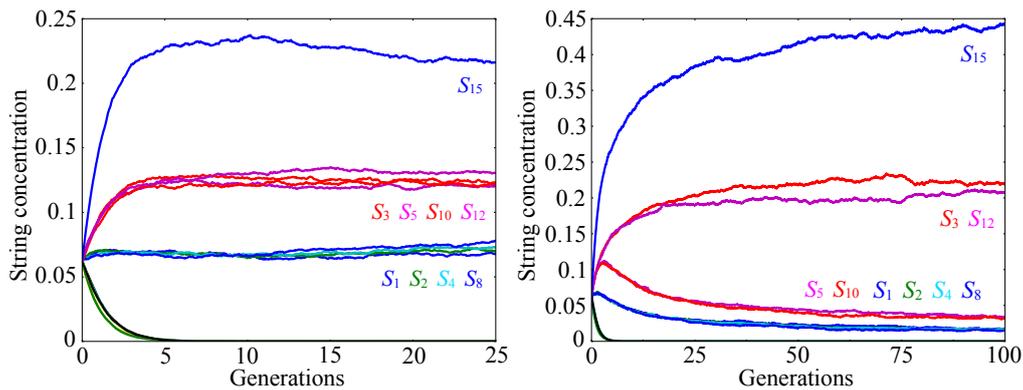


Figure 4.16: Only the destructor was not allowed to replicate in this system. (a) left, S_X-F_{ran} , $M = 10^5$, 25 generations. (b) right, S_X-F_{sel} , $M = 10^5$, 100 generations.

Destructor [0000] and exploiter [1111] production prohibited

Each of the six systems in the following section were initialized with a random string population. The filters that were applied in this section were the inhibition of destructor and exploiter creation. All reactions that would

produce destructors or exploiters are considered not to take place and their iteration ‘turns’ are restarted. Figs. 4.17–4.19 show six simulations each initialised with a heterogeneous population of 10^5 strings. Each system utilized a different folding mechanism. Subgroups in Figs. 4.17 and 4.19 took slightly longer reach their final concentrations than in the previous section.

Figs. 4.17.a and 4.17.b show two systems, each initialised with all strings. Fig. 4.17.a used canonical folding and Fig. 4.17.b used transposed canonical folding. Both systems displayed very similar behaviour: a subgroup of four strings that contain single one-bits, S_1 , S_2 , S_4 , and S_8 , dominates the population and reaches 23% after 25 generations.

In Fig. 4.17.a, one-bit subgroup members occupy on average 23.8% each, and combined they account for more than 95% of the population. In Fig. 4.17.a the only other surviving subgroup contains strings with two one-bits each, S_3 , S_5 , S_{10} , and S_{12} . The two-bit subgroup members each occupy 1.2% on average, and make up for almost 5% of the population.

In Fig. 4.17.b, one-bit subgroup members occupy on average 23.7% each, and combined they account for almost 95% of the population. In Fig. 4.17.b the other surviving subgroup contains the same strings as Fig. 4.17.a, which two one-bits each, but in Fig. 4.17.b the group is split into two subgroups, S_5 and S_{10} , which average 1.8% each, and S_3 and S_{12} , which average 0.8% each. The two-bit subgroup members make up just over 5% of the population.

Fig. 4.18.a used topological folding and Fig. 4.18.b used transposed topological folding. These two systems displayed different behaviour from each other and from the previous two systems. In Fig. 4.18.a the subgroup of S_1 , S_2 , S_4 , and S_8 dominate the population and average 18.6% each. The other surviving subgroup in Fig. 4.18.a is the remainder of the two-bit subgroup,

4.4. Heterogeneous Systems

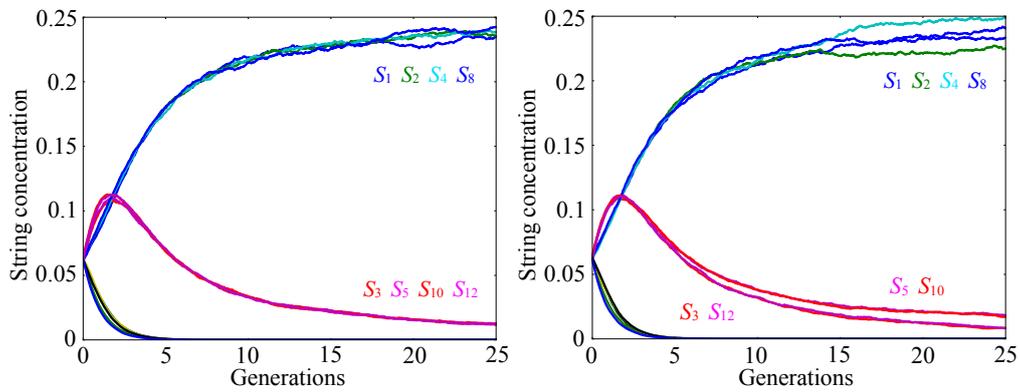


Figure 4.17: The destructor and exploiter were not allowed to replicate in this system. (a) left, S_X-F_{can} , $M = 10^5$, 25 generations. (b) right, $S_X-F_{t.can}$, $M = 10^5$, 25 generations.

containing S_5 and S_{10} , which average 12.9% each.

Two subgroups are able to survive in Fig. 4.18.b, half of the two-bit subgroup containing S_3 and S_{12} , and the one-bit subgroup containing S_1 , S_2 , S_4 , and S_8 . S_3 and S_{12} average 23% each, while the one-bit subgroup members each average 13.5%.

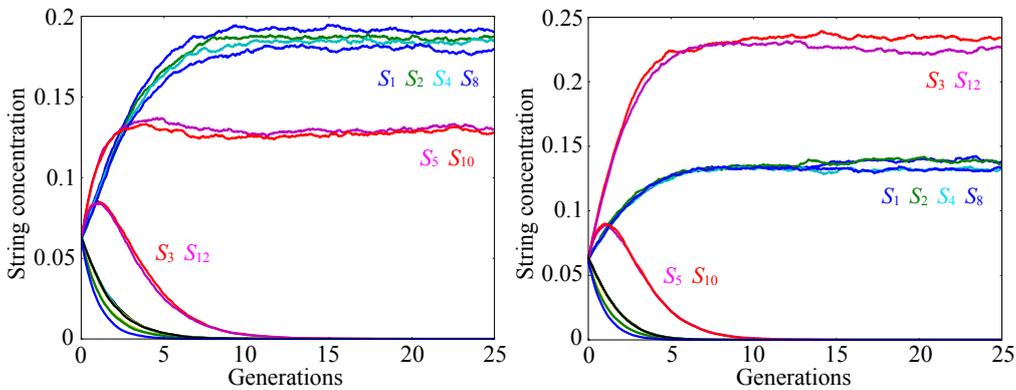


Figure 4.18: The destructor and exploiter were not allowed to replicate in this system. (a) left, S_X-F_{top} , $M = 10^5$, 25 generations. (b) right, $S_X-F_{t.top}$, $M = 10^5$, 25 generations.

Fig. 4.19.a used random folding and Fig. 4.19.b used selective folding. Fig. 4.19.a displays similar initial behaviour with Fig. 4.17.b: the one-bit

4.4. Heterogeneous Systems

string group, which average 23.5% each, dominates the population, followed by a split two-bit subgroup, S_5 and S_{10} , and S_3 and S_{12} . The split in the two-bit subgroup is more pronounced from earlier in the system, and after 25 generations S_5 and S_{10} have reached 2%, and S_3 and S_{12} have reached 1%.

Fig. 4.19.b is similar to Fig. 4.19.a for the first five generations, after that the system changes behaviour and enters a unique state. Initially the one-bit string group dominates the population, followed by the split two-bit subgroup. However, the one-bit subgroup does not stay unified, and S_1 and S_4 gain advantage over S_2 and S_8 , which begin to drastically decline after 50 generations. S_1 and S_4 go on to dominate the population and after 100 generations they average 47.6% each. S_5 is not afflicted by S_1 or S_4 and it preserves itself at 2.7% through passive replication from S_1 and self-replication. Three subgroups cling to existence below 1% each: S_2 and S_8 (0.7%), S_3 and S_{12} (0.3%), and S_{10} (0.04%).

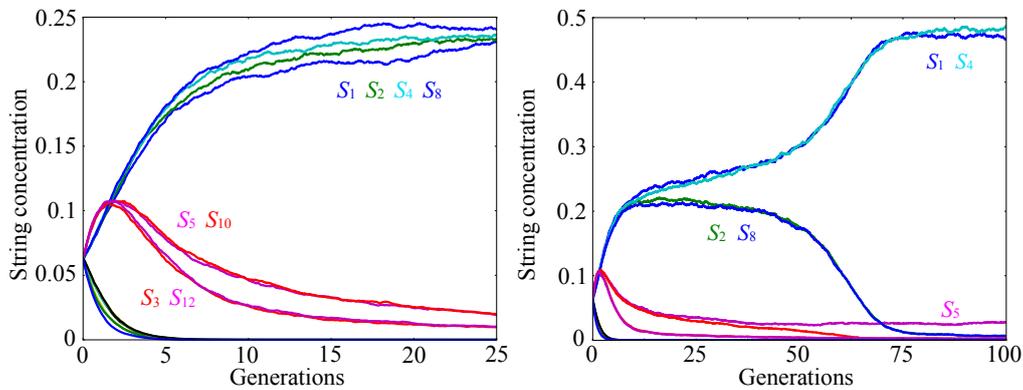


Figure 4.19: The destructor and exploiter were not allowed to replicate in this system. (a) left, S_X-F_{ran} , $M = 10^5$, 25 generations. (b) right, S_X-F_{sel} , $M = 10^5$, 100 generations.

Destructor [0000] production prohibited; highly specific substitution, $n=100$

Each of the six systems in the following section were initialized with a random string population. The only filters applied in this section are the inhibition of destructor string creation and the probability of substitution. All reactions that would produce destructors are considered not to take place and their iteration ‘turns’ are restarted. In addition, once each iteration, a random string from the population is examined for substitution. If the string fails the substitution test it is replaced by a copy of a random member of the population. In this section the substitution constant is set to $n = 100$, which yields constant exploiter substitution and a negligible chance of ‘normal’ string substitution. This section slightly resembles the previous section, as diversification of subgroups took relatively long. Figs. 4.20–4.22 show six simulations each initialised with a heterogeneous population of 10^5 strings. Each system utilized a different folding mechanism.

Fig. 4.20.a used canonical folding and Fig. 4.20.b used transposed canonical folding. Both systems displayed similar behaviour: the exploiter string is subdued while the one-bit subgroup thrives and dominates the two-bit subgroup. One-bit subgroup members, S_1 , S_2 , S_4 , and S_8 , occupy on average 22.6% each, and combined they account for over 90% of the population. Two-bit subgroup members, S_3 , S_5 , S_{10} , and S_{12} , occupy on average 2.3% each, and make up almost 10% of the population. The exploiter, S_{15} , is severely suppressed, and in both systems it manages to stay just above 0.1%.

Fig. 4.21.a used topological folding and Fig. 4.21.b used transposed topological folding. These two systems displayed different behaviour from each

4.4. Heterogeneous Systems

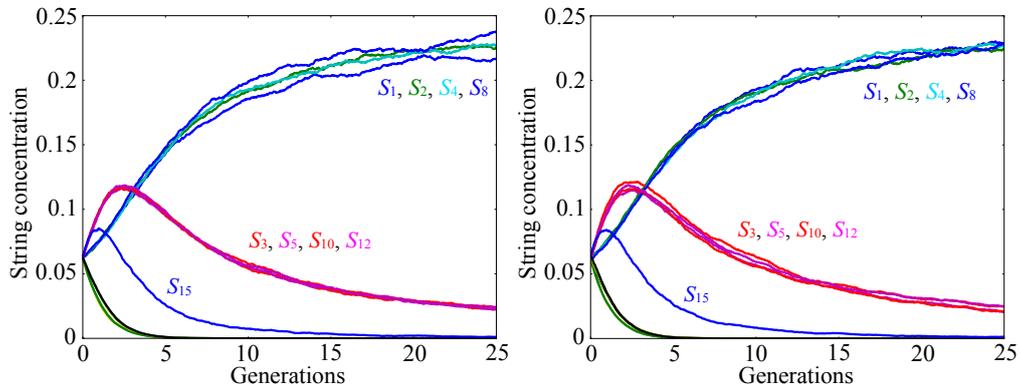


Figure 4.20: The decay constant, n , was set at 10^2 . (a) left, S_X-F_{can} , $M = 10^5$, 25 generations. (b) right, $S_X-F_{t.can}$, $M = 10^5$, 25 generations.

other and from the previous two systems. In Fig. 4.21.a the one-bit subgroup, S_1 , S_2 , S_4 , and S_8 , dominate the population and each average 17.9%. They are closely followed by two members of the two-bit subgroup, S_5 and S_{10} , which average 14.2% each. The remainder of the two-bit subgroup does not survive, and the exploiter string is suppressed until its demise.

Only two groups of strings are able to survive in Fig. 4.21.b, half of the two-bit subgroup containing S_3 and S_{12} , and the one-bit subgroup, S_1 , S_2 , S_4 , and S_8 . The two-bit subgroup plateau at an average of 32.3% each, while the one-bit subgroup average 8.8% each.

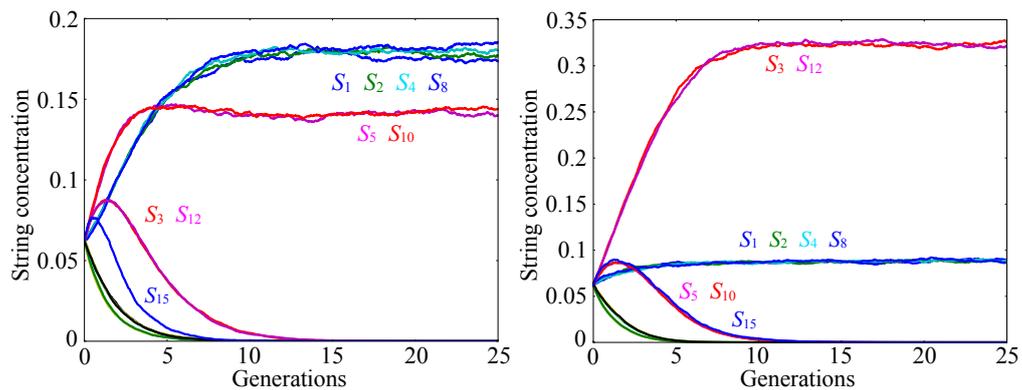


Figure 4.21: The decay constant, n , was set at 10^2 . (a) left, S_X-F_{top} , $M = 10^5$, 25 generations. (b) right, $S_X-F_{t.top}$, $M = 10^5$, 25 generations.

4.4. Heterogeneous Systems

Fig. 4.22.a used random folding and Fig. 4.22.b used selective folding. Fig. 4.22.a shares very similar initial behaviour with Fig. 4.25.a: the one-bit subgroup dominates the population, followed by a split two-bit subgroup, S_5 and S_{10} , and S_3 and S_{12} . The split in the two-bit subgroup is very pronounced from early in the system. The strings in the one-bit subgroup average 22.1% each by the 25th generation. The subgroup of S_5 and S_{10} reach 3.7% by the 25th generation, and S_3 and S_{12} average 2.0% each by the 25th generation.

Fig. 4.22.b is not similar to any other system. For a long period two sub-strings, S_2 and S_8 , dominate the population, followed closely by another two sub-strings, S_1 and S_4 , while S_5 and S_{10} linger at the bottom of the concentration range. However, after the 800th generation, the two thriving groups swap positions, and S_1 and S_4 enter an stochastic period of competition by passive replication. They average 48.6% while S_5 continues unaffected at 2.3%.

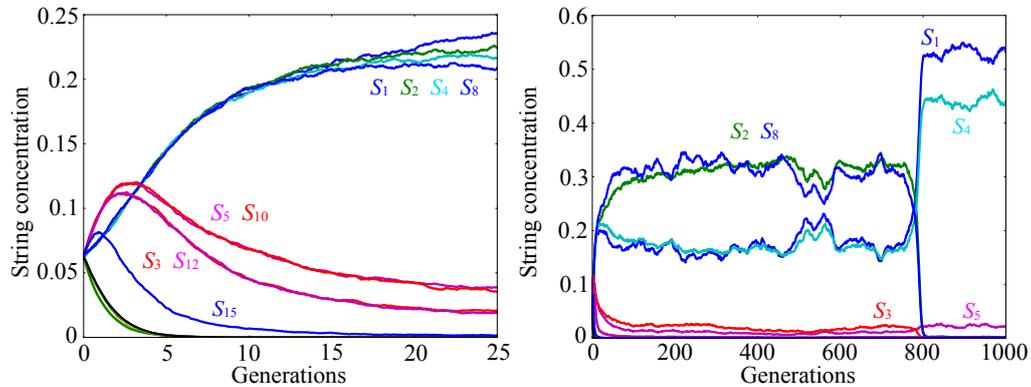


Figure 4.22: The decay constant, n , was set at 10^2 . (a) left, S_X-F_{ran} , $M = 10^5$, 25 generations. (b) right, S_X-F_{sel} , $M = 10^5$, 1000 generations.

Destructor [0000] production inhibited; proportionally specific substitution, $n=1$

Each of the six systems in the following section were initialized with a random string population. The only filters applied in this section are the inhibited of destructor string creation and the probability of substitution. Once each iteration a random string from the population is examined for substitution. If the string fails the substitution test it is replaced by a copy of a random member of the population. In this section the substitution constant is set to $n = 1$, which still yields constant exploiter substitution, but the chance of ‘normal’ string substitution is no longer negligible, and the chance for substitution linearly increases with the concentration of 1-bits in the examined string. In this section diversification of subgroups occurs relatively quickly, and in all systems, except Fig. 4.25.b, subgroups are quenched by the substitution filter. Figs. 4.23–4.25 show six simulations each initialised with a heterogeneous population of 10^5 strings. Each system utilized a different folding mechanism.

Fig. 4.23.a used canonical folding and Fig. 4.23.b used transposed canonical folding. Both systems displayed almost identical behaviour: the subgroup containing one-bit strings thrives, while the exploiter and the two-bit subgroup strings are suppressed. After 25 generations the one-bit strings average 24.95%, and the strings from the two-bit subgroup still exist, but they only average 0.05%. The main visible difference between these two systems is the branching out of one-bit subgroup members in Fig. 4.23.b, opposed to the even growth in Fig. 4.23.a. If the systems in Fig. 4.23.a and Fig. 4.23.b are run for more generations, strings from the two-bit subgroup disappear from the system, and the only strings from the one-bit subgroup remain. In Fig. 4.23.b S_1 and S_8 fluctuate between 20%–30% each, while

4.4. Heterogeneous Systems

S_2 and S_4 remain closer to 25%. Whenever S_1 or S_8 vary towards 5% in either direction they are brought back towards 25% by the action of other strings. Only in systems with smaller populations do S_1 or S_8 manage to individually dominate populations.

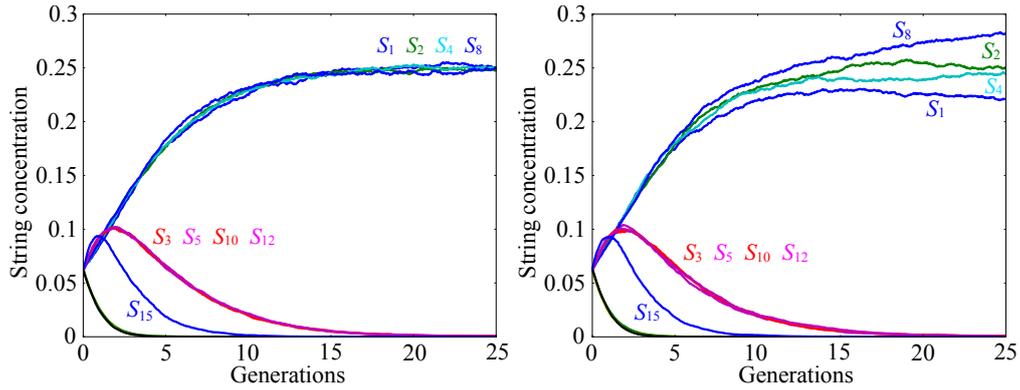


Figure 4.23: The decay constant, n , was set at 10^0 . (a) left, S_X-F_{can} , $M = 10^5$, 25 generations. (b) right, $S_X-F_{t.can}$, $M = 10^5$, 25 generations.

Fig. 4.24.a used topological folding and Fig. 4.24.b used transposed topological folding. These two systems shared similar characteristics with each other displayed behaviour similar to the previous two systems. In both systems the subgroup containing one-bit strings survives, while the exploiter and the two-bit subgroup strings are suppressed. The main difference between the Figs. 4.23.a and 4.23.b and Figs. 4.24.a and 4.24.b is the split in the two-bit subgroup before the decline of its members. In Fig. 4.24.a S_3 and S_{12} are the first two-bit strings to decline, and are no longer part of the population by the 10th generation. S_5 and S_{10} are more successful than S_3 and S_{12} , but they are also unable to survive in the long term, and they leave the population before the 25th generation. In Fig. 4.24.b S_5 and S_{10} are the first two-bit strings to decline. The exploiter string initially performs better than Fig. 4.24.a, but it is too suppressed, and it fades from

4.4. Heterogeneous Systems

the population before S_5 and S_{10} . Although S_3 and S_{12} are initially more successful than S_5 and S_{10} , they are also unable to survive in the long term, and they are eventually forced out of the system.

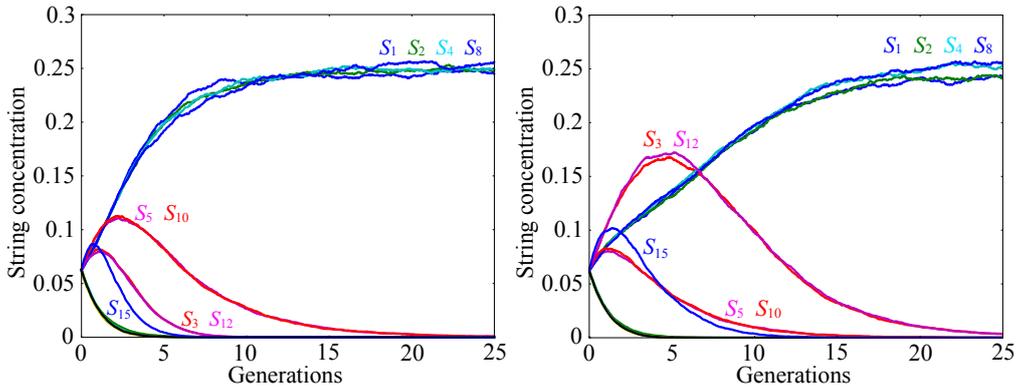


Figure 4.24: The decay constant, n , was set at 10^0 . (a) left, S_X-F_{t-top} , $M = 10^5$, 25 generations. (b) right, S_X-F_{t-top} , $M = 10^5$, 25 generations.

Fig. 4.25.a used random folding and Fig. 4.25.b used selective folding. Fig. 4.25.a displays behaviour similar to that of the first two systems, combined with the previous two systems. The exploiter and the two-bit subgroup are suppressed and do not survive in the long term. There is a slight split in the two-bit subgroup between the third and 15th generations, with S_5 and S_{10} briefly leading S_3 and S_{12} , but this does not stop their eventual demise, leaving the one-bit subgroup as the only surviving subgroup.

Fig. 4.25.b is similar to Fig. 4.22.b, where the one-bit subgroup eventually dominates the population, until internal competition splits it into two subgroups. The main difference between this system and Fig. 4.22.b is the suppression of the two-bit strings, and the initial slight dominance of S_2 and S_8 over S_1 and S_4 . After an initial plateau around 100 generations S_1 and S_4 begin to overpower S_2 and S_8 , and soon after 200 generations S_2 and S_8 fade from the population.

4.4. Heterogeneous Systems

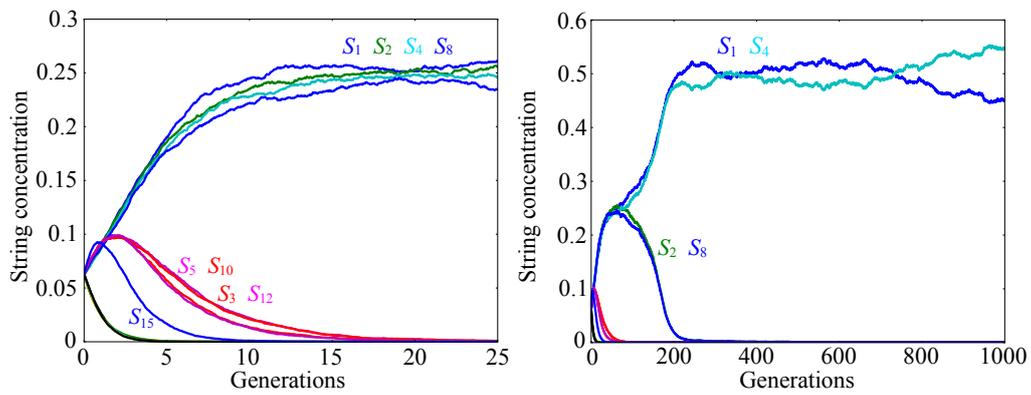


Figure 4.25: The decay constant, n , was set at 10^0 . (a) left, S_X-F_{ran} , $M = 10^5$, 25 generations. (b) right, S_X-F_{sel} , $M = 10^5$, 1000 generations.

Chapter 5

Discussion

This thesis gave a broad overview of what ACs are and how they are useful, a brief review of several types of ACs and their applications, and an in depth analysis of one specific type of AC: the four-bit binary string system. Using the Python programming language, we recreated the model designed by Banzhaf [1, 2] for *in silico* examination. The results from our models were verified against the results from Banzhaf's [1, 4] models. Both systems produced the same data sets when benchmarked against each other for comparison (Sections 4.2 and 4.3 vs. [1, 4]). We extended their model further to be able to incorporate random and selective folding types, and to be able to exclude exploiter strings in addition to destructors. While excluding the exploiter may be a constraint, because it is an element being removed from the system, it provides population space for other strings to thrive, however, the inclusion of random and selective folding types increases the emergent potential of the system by simultaneously allowing more types interactions to occur.

The initial motivation was to identify an existing AC that could be used to elucidate the sequence-function relationships in objects with multi-

ple states, and how the mapping of these nominable entities (objects that can only be described by naming their sequence [10]) between different states affects the function of the mapped object. Specifically, we wanted a system analogous with the DNA–RNA–protein system, with emphasis on objects that can occupy multiple states, i.e., objects with ‘sequential’ and ‘functional’ states. The interest in sequence-function relationship stems from its importance for self-production, which is a necessary precursor for self-replication. The initial investigation of the sequence-function relationship and self-replication lead to the broadening of the study to include self-organization in artificial systems, as it was shown to be a crucial component of self-perpetuating organizations.

Chapter 2 gave a brief review of ACs, from their mental conception to their physical implementation, and their current applications. The goal of this section was to give an overview of what ACs are, how we design and differentiate between them, and what practical applications they have. This section also contains several examples of which ACs we initially investigated, and why we eventually chose to investigate binary string systems.

Alan Turing was very interested in the organization of living entities, and he was one of the first people to formally conceptualize the logic of life [45]. The ‘Turing machine’ was a universal and pliable concept, and it paved the way for the study of artificial intelligence, artificial life, and artificial chemistries. The ‘Turing machine’ also possessed all three properties needed for modernly defined ACs: A set of objects (S) that can be manipulated, a set of rules (R) that define how objects interact, and an algorithm (A) that that applies the reaction rules and drives reactions between objects. All current ACs can be described by this triple (S, R, A), and its is almost impossible to create an AC that cannot be described by

these three properties.

Three other properties are also crucial when identifying ACs and designing AC systems. Systems are divided along three axes: analogous or abstract, constructive or non-constructive, and implicit or explicit. These properties of the system depend on the (S, R, A) properties.

- Analogous systems are based on real processes, or they at least have a theoretical counterpart, while abstract systems do not need to be related to anything real. Analogy and abstraction depend on the objects and reaction rules.
- Constructive systems are able to generate new objects, while non-constructive systems are limited by their given object set. System construction potential solely depends on the reaction rules, as they govern object interactions and object production.
- Systems are also either explicit or implicit, based on their mechanism of interaction: a reaction is implicit if depends on the structure of the object, and a reaction is explicit if the result is user defined or based on structureless objects.

Although Turing did not know it, the system he conceptualized was analogous to the DNA–RNA–protein system. His system was also constructive, as the machine was able to modify and add to the existing information ‘tape’ or sequence information. The Turing machine also had both implicit and explicit mechanisms: on a level of local interaction the reactions were explicit, as they were user defined, but on the mesoscopic level the entire reaction mechanism was implicit, as the outcome is dependent on entire sections of interpreted ‘tape’, making the reaction non-deterministic and overall implicit.

There were two other types of ACs that, because of their design and properties, could have been investigated in the present study. These systems, binary string automata and typogenetic systems, are also analogous to the DNA–RNA–protein organisation, and they are useful when investigating sequence–function relationships and self-organization phenomena. The main algorithm for both of these ACs is the same: objects composed of character sequences are mapped and interpreted as ‘sequence of instructions’ operators, which are capable of operating on themselves or on other character sequences, altering them and changing the function of their respective operational forms.

The system we chose to investigate and which is described and analysed in Chapter 3 was the four-bit binary string system. It has a simple design and implementation, can yield complex results from interactions between a small group of objects, and is analogous to the DNA–RNA–protein organisation; these properties makes the system useful for investigating the relationship between sequence and function in objects that have alternative forms. The implicit reaction mechanism, which relies on the structure of the interacting objects, preserves locality during object interactions and allows the four-bit binary system to express constructive and emergent behaviour, such as the introduction of new objects through existing object interactions, and the self-organization of entire populations through random object interaction. The four-bit binary string system was chosen over other matrix multiplication systems, such as longer length or variable length systems, because it has the smallest non-trivial length strings and because of its ability to display behaviour similar to larger, more complex, string systems, while retaining simple design and implementation.

Chapter 3 provided an introduction to binary sequences and programs,

and the ways in which programs operate on and transform data. Here the analogy between biological catalysts (specifically enzymes and ribozymes) and chemical catalysts, and software programs, was made. The shorthand writing form for strings was described next, accompanied by a table containing the structure and shorthand name of each string in the four-bit system. In Section 3.1 mapping of strings into operators was discussed, with examples of the four standard mappings, two ‘non-biological’ mappings, and two combinatorial mappings (random and selective folding). In Section 3.2 the reaction rules that govern binary string interactions were discussed, and in Section 3.3 the resulting five reaction types were examined: new string production from dissimilar interacting strings, new string production from a self-reaction, self-replication, active replication, and passive replication. These five reaction types were further generalized into three reaction classes, new string generation, self-replication, and replication reactions, that were used in describing the reaction tables in Section 4. Section 3.4 compared our model with the original, designed by Banzhaf [1, 2]. The population dynamics depended on the reaction vessels in which string populations exist, and how excess strings are dealt with by random deletion, string consumption, or operator consumption; of great importance was the way in ‘lethal’ strings, the destructor and exploiter, was dealt with, primarily by exclusion through elasticity, or through a substitution probability (only for the exploiter).

Chapter 4 reported the results of the investigation, using the model designed in Section 3.4. Section 4.1 described the four reaction tables and their contents. There were 256 different reactions per reaction table, and each table had its own pattern of string formation. The canonical and topological folding types shared the same number of reactions per major reaction

class. The transposed-canonical folding type had eleven self-replication reactions, the most of all folding types, and the transposed-topological form had the most strings belonging to the type two reaction class of new string generating self-reactions. The reaction tables also produced the exact same number of string species per reaction table, and strings from structurally related subgroups were produced in the same numbers across reaction tables. The destructor and exploiter were respectively the most prolific strings, followed in order by the most of the two-bit subgroup (S_3 , S_5 , S_{10} , and S_{12}), the one-bit subgroup, the three-bit subgroup, and the remainder of the two-bit subgroup (S_6 and S_9). Section 4.1 also described the various operators that can be formed from all folding types (vertically, horizontally, and diagonally symmetrical, as well as one-bit and three-bit operators) and the types of strings they are likely to produce.

Various systems were then analysed, namely: (1) simple homogeneous systems that were initialised with only one type of string, more specifically a string the self-reaction of which produces a new string. We specifically chose to examine systems that did not lead to the immediate production of destructors or exploiters; (2) the S_6 and S_9 systems using random and selective folding instead of standard folding; (3) heterogeneous systems that were initialized with all string species. Four groups of six systems were investigated: each group used a different type of substitution method for dealing with lethal strings, each system was initiated with all string species, and systems within groups each utilized one of six different folding methods.

Section 4.2 described the analysis of several simple homogeneous systems, which were initialised with one string type, and used one folding type throughout each simulation. Of the S_6 , S_7 , S_9 and S_{11} strings, S_6 and S_9 generated the highest number of systems that fulfilled our selection criteria

of new-string generating self-reactions. Because of the complexity of the metabolic diagrams and the difficulty encountered while trying to design visually sensible metabolic diagrams, only the first few systems were also depicted as metabolic diagrams. For the rest only graphs and metabolic tables were used. In the systems where metabolic diagrams were used they clearly illustrated the underlying logic of system states and the functional organizations that may emerge. The S_7 canonical folding system in Figs. 4.1a and 4.1b was an example of a dead end system where all initial strings were eventually metabolised into a single other string species. The S_7 self-reaction produced the exploiter S_{15} , and all subsequent reactions between S_7 and S_{15} produced S_{15} , leading to a static population of self-producing S_{15} strings. Two other simple homogeneous systems, the S_9 transpose topological folding system (Figs. 4.2a and 4.2b) and the S_{11} topological folding system (Figs. 4.4a and 4.4b) always deterministically reached their respective dead-end states, with S_9 being metabolised into S_5 , and S_{11} being metabolised into S_{15} through S_7 as an intermediate species. The remaining simple homogeneous systems could be grouped into two categories based on their end states: systems with stable end states, or systems with stochastic end states. The S_9 topological folding system (Figs. 4.5a and 4.5b) and the S_6 transposed topological folding system (Figs. 4.7a and 4.7b) were examples of systems with stochastic end states. Stochastic systems had several characteristic features, which commonly included containing operators that promote passive and self-replication, and other operators that only had elastic reactions with remaining strings and were in effect completely inert. The outcome of stochastic systems was indeterminable and the result depended on the random selection mechanism. The S_6 canonical (Figs. 4.3a and 4.3b) and transposed topological folding systems (Figs. 4.7a and 4.7b) were ex-

amples of systems with stable end states. Systems with stable end states had their own characterising features, and in both of these systems all self reactions only produced one of the species present, while the other string was produced by a combination of passive and active replication reactions. The resulting organization also functioned as a negative feedback loop, in which strings were prevented from diverging too far from each other while their concentrations were being equalized.

In Chapter 4.3 we examined the S_6 and S_9 systems using the random and selective combinatorial folding types instead of a single folding type. S_6 and S_9 were chosen for investigation because they formed the most developed organizations out of all of the simple homogeneous systems; both of these strings led to systems that were capable of developing into stochastic or stable end states. It therefore followed that S_6 and S_9 should show complex system development and network formation if combinatorial folding methods were to be applied to them. The number of possible interactions between objects in combinatorial folding systems made the use of metabolic diagrams and reaction tables impractical. Therefore, only partial metabolic tables containing reactions of the initial string were shown. Interestingly, in both S_6 and S_9 systems the reactions of the initial string with the strings it subsequently produced led to the formation of closed subsets in both random and selective systems. This means that all strings in both systems could be accounted for by the interaction of the initial string with any other string, i.e., no new strings were added to the population by the interaction of progeny strings alone. The reactions in the S_6 random folding system (Figs. 4.9a and 4.9b) led to the production of ten other strings. These 11 strings could be divided into three groups and one pseudo-group. S_6 and S_9 were the first group to perish, because there were not enough reactions

from other strings that produce S_6 or S_9 , and not enough strings that S_6 or S_9 could interact with to actively or passively replicate. Indeed, S_6 and S_9 only occurred twice per reaction table, as indicated in Table 4.2, and they were not able to survive in the S_6 random system. The next two groups that faded from the system were (S_{10}, S_{12}) and (S_2, S_4, S_8) . These groups were unable to cope with the exploiter string as they did not have enough reactions with the exploiter that produced them. The (S_1, S_3, S_5, S_{15}) group survived because of the beneficial relationships that S_3 and S_5 had with S_1 , which was capable of utilizing the exploiter to the advantage of S_3 and S_5 . The downfall of the group, in the given system, could be attributed to S_5 not contributing to the S_1 species; S_5 produced S_1 through active replications with S_1 and S_3 , but it would have balanced the remaining organization through contributing to the S_1 population, primarily through passive replications with itself, S_1 , or S_3 . This was not the only possible outcome of the system, and the resulting organization of strings $[(S_{15})(S_3, S_5)(S_1)]$ was capable of exhibiting multiple states. The organization had at least three states, and it was capable of autonomously switching between these states. The most stable state often occurred in larger systems (ca. 10^6). The two other states were variations of the first state; they were inverse states of each other, and more likely to occur in systems with a population of less than 10^6 strings. They were characterised by a dominance of either S_1 or S_{15} for a prolonged period. If the system became partially dominated, it was capable of regressing back to the stable state, or progressing further into a single string system.

All selective systems that we examined essentially began as random folding systems. This is because, in selective systems, all folding types must start with an equal chance of being selected in the beginning of the sim-

ulation. Because both random and selective systems were allowed to use any folding type, when two systems were initiated with the same subset of strings, but one system used random folding and the other system used selective folding, they would always produce the same resulting subset of strings. The resemblance in initial development of random and selective systems was uncanny; comparing Fig. 4.9a with Fig. 4.10a provides a clear example of initial growth similarity. The main difference between Fig. 4.9a and Fig. 4.10a was the improved growth of three strings, S_3 , S_{12} and S_{15} , and the slight decline in growth of all other strings. The long term differences were much more apparent, as can be seen by comparing Fig. 4.9b with Fig. 4.10b. In Fig. 4.9b, after 2000 generations, the system still had not reached a stable or end state, whereas in Fig. 4.10b the system had adapted and already reached a stable state before 200 generations had passed. The survival of the resulting organization of three strings could largely be attributed to two factors: the passive replication of all strings in the resulting organization, and the ability of S_3 and S_{12} to adapt to advantageous folding types. Coincidentally, the folding types that S_3 and S_{12} adapted to also caused their respective operators to promote passive replication within the remaining organization, which decreased the number of new strings that were added to the population, and increased the amount of S_3 , S_{12} and S_{15} . The ability of S_3 and S_{12} to adapt to the exploiter resulted in a stochastic end system of passive replicators.

The reactions in the S_9 random folding system (Figs. 4.12a and 4.12b) led to the production of only four other strings. These four strings form a single group that remained indefinitely stable in larger systems. This same group of four strings was unable to sustain itself (Fig. 4.9b). The group survived because of the beneficial relationship between (S_3 , S_5) and

S_1 , which was capable of utilizing the exploiter to the advantage of S_3 and S_5 . The downfall of the group could occur in systems of strings with a population of less than 10^6 strings, where S_1 and S_{15} were more capable of reaching the threshold concentration than systems of 10^6 or larger. This group is discussed extensively in Section 4.2, under 0110 random folding, and in the discussion of the S_6 random folding system earlier only in this chapter. In short, the resulting organization of strings, $[(S_{15})(S_3, S_5)(S_1)]$, was capable of autonomously switching between least three states. The most stable state occurred in larger systems (10^6), while the two other states were inverse states of each other and were more likely to occur in systems with a population of less than 10^6 strings and were characterised by a dominance of S_1 or S_{15} . Where the system became partially dominated by S_1 or S_{15} , it was capable of regressing back to the stable state, or progressing further into a single string system containing either S_1 or S_{15} .

The similarity in initial development between random and selective systems is clear when the first five generations of Fig. 4.12a and Fig. 4.13a are compared. However, when Fig. 4.12b and Fig. 4.13b are compared, the difference in long term system behaviour is immediately apparent. The main difference between Fig. 4.12b and Fig. 4.13b is the clear division of one group into two competing groups. The division, and subsequent competition, between the two groups is a consequence of selection for active replication: S_1 and S_5 , and S_3 and S_{15} , coincidentally selected reactions that produce each other in equal ratios. The different behaviour of the strings from the random systems changed the states that the organization of strings were capable of producing. Previously, one organization could enter one of three states, and it was capable of progressing into one of two single string dominant states. In Fig. 4.13b the organization divided into

two smaller organizations, which competed for space in the population. After 2000 generations, the system still hadnot reached a stable or end state, but the organization containing S_3 and S_{15} was on its way to dominating the population. The survival of the smaller organizations could be attributed to coincidental selection for reactions that produced members of the same organization.

In Section 4.4 we examined four groups of six systems: each group used a different type of substitution method for dealing with lethal strings, each system was initiated with all string species, and systems within groups each utilized one of six different folding methods. Throughout the entire section on heterogeneous systems only nine of the 15 strings (destructor excluded) featured prominently. These strings could be roughly divided into three groups of strings: the exploiter, the group of one-bit strings (S_1, S_2, S_4, S_8), and the group of two-bit strings (S_3, S_5, S_{10}, S_{12}). On occasion the two larger groups will split into two consistent smaller groups: $(S_1, S_2, S_4, S_8) \rightarrow (S_1, S_4)(S_2, S_8)$ and $(S_3, S_5, S_{10}, S_{12}) \rightarrow (S_3, S_{12})(S_5, S_{10})$.

Fig. 4.14a, Fig. 4.14b, and Fig. 4.16a are good examples of an organization that was commonly encountered in heterogeneous systems, and sometimes arose from homogeneous populations. At a glance it appears as though this organization is characterized by dominance of the exploiter, followed by the two-bit and one-bit groups respectively. However, if dominance is measured with group weight, instead of individual weight, then the two-bit group was the most dominant group, occupying 50% of the population. The combined weight of the one-bit group (27%) also slightly exceeded the concentration of the exploiter string (23%), placing the one-bit group ahead of the exploiter. There was also no direct relationship between the exploiter and the one-bit group; there were no replication reactions

between the exploiter and any of the one-bit strings, and all reactions between the exploiter and a one-bit string produced one of the two-bit strings. Being produced by reactions between other groups, and the ability to replicate when interacting with the exploiter (which still has the single highest string concentration) and single-bit strings, placed the two-bit string group comfortably in the middle of two competing groups. This effectively left an organization where one group operated as a buffer between two competing groups to stabilize the organization.

The organization that appears in Fig. 4.15a was a variant of the previous organization described. Small changes in local interactions, such as the change of intergroup behaviour and the formation of stable internal organizations, accumulated and caused the weight of the different groups to change. Changes in this group led to the dominance of the one-bit group, with a combined weight of 40%, the increased concentrations of (S_5, S_{10}) , and the decreased concentrations of S_{15} and (S_3, S_{12}) . However, this organization remained functionally similar to the previous organization, because the two-bit strings still acted as a buffer group between the one-bit group and the exploiter, and there still were no replication reactions between the one-bit group and exploiter string. The only organizational changes were the slight collapse in concentration of the exploiter and the (S_3, S_{12}) subgroup.

The organization that appears in Fig. 4.15b was very different from the previously described organizations, as it only contained three strings: S_3 , S_{12} , and the exploiter, S_{15} . The resulting system was entirely composed of passive-replicators, which produced an unstable organization with a stochastic outcome. Systems with populations consisting of only passive-replicators always produced the same type of organization, where the out-

come of the system, and dominance of individual strings, was entirely stochastic, regardless of the number of different passive-replicators in the population.

The resulting organization in Fig. 4.16b was similar to the organization in Fig. 4.15b, with the addition of six weaker strings. The selective folding mechanism allowed the same three strings that dominated the Fig. 4.15b to be dominant in this system, but the additional folding possibilities also allowed interactions to continuously produce ‘by-product’ strings. These ‘by-products’ were not able to compete on their own with the three dominant strings, and their survival was dependant on non-replicative interactions between the dominant strings. If one of the of three dominant passive-replicators did not survive because of the stochastic nature of the system, the ‘by-product’ strings that depended on its replications faded from the system as well.

In Figs. 4.17a–4.19b the exploiter and the destructor strings were not allowed to reproduce. In all of the systems, except in Fig. 4.18b, this had the effect of crippling the success of the two-bit group, and the one-bit string group was able to flourish in its place. The suppression of the exploiter had a clear influence on the success of the two-bit group, and the loss of beneficial relationships between the two-bit group and the exploiter were to the advantage of the one-bit group.

In Fig. 4.17a, Fig. 4.17b, and Fig. 4.19a, the one-bit group dominated the system. The two-bit groups in these systems slowly declined because of random passive and active replications between members of one-bit and two-bit groups. Over time the two-bit group was forced out of the system and the one-bit group was left as the dominant organisation. Fig. 4.19b followed the same initial path towards being dominated by the one-bit group, but

over time selection favoured the smaller organization of S_1 and S_4 . S_5 was mostly unaffected by S_1 and S_4 because it passively replicated when they operated on it.

The situation was slightly different in Figs. 4.18a and 4.18b, and in both systems a group of two two-bit strings survived alongside the one-bit organization. In Fig. 4.18a the two-bit strings were a stable organization and survived by passive replication, while in Fig. 4.18b the two-bit strings formed an unstable organization that survived through active replication. The two-bit strings in Fig. 4.18b did better than those in Fig. 4.18a because they participated in more reactions that were either elastic or were beneficial for themselves, and in fewer reactions that were beneficial for the one-bit string group.

There were no new types of organizations that appeared in Figs. 4.20a–4.25b. In these systems the threshold value for string substitution was tested at $n = 1$ and at $n = 100$, simulating a system very loosely substituting strings based on the fraction of one-bits in the string ($n = 1$), and a system very strict on substituting only the exploiter ($n = 100$). In Figs. 4.20a–4.22b the only effects that the introduction of the substitution probability had were the altered the growth rates and final concentrations of string groups. Unlike the systems in Figs. 4.17a–4.19b, where reactions that produce exploiters were elastic and were not allowed to take place, the systems in Figs. 4.20a–4.22b were capable of producing exploiters, but the highly specific substitution probability ($n = 100$) suppressed the exploiter out of the systems in each scenario. In Figs. 4.23a–4.25b the substitution constant n was set to 1, giving all strings a chance to be substituted based on the fraction of one-bits they contained, with each one-bit increasing the likelihood of substitution by 25%. Throughout Figs. 4.23a–4.25b the one-bit

string group was the only group able to maintain itself as an organization, and in all $n = 1$ systems, except Fig. 4.25b, all four one-bit strings were able to survive together. In Fig. 4.25b, just as in Figs. 4.19b and 4.22b, S_1 and S_4 were able to select for themselves and against S_2 and S_8 , and they were able to dominate the population together.

In this study we set out to elucidate the sequence–function relationship by using artificial objects that can be mapped back and forth between multiple states. The mapping of these objects between different states affects the function of the mapped object. When a population of objects are allowed to interact, the phenomena of self-production and self-organization present themselves [27]. We examined multiple systems with different parameters and were able to identify several types of self-sustaining organizations. The survivability of an organization is dependent on the interactions of its components, and the components must be capable of continuously producing themselves if they are to maintain the organization. The ability of the objects to maintain themselves depends of their function, which is determined by their mapping. It was shown that self-maintaining organizations are possible even in systems with only one folding type, and robust organizations that survive in systems with multiple folding types are capable of self-maintenance as well.

5.1 Critique

One of the primary objectives of this project was to study the relationship between sequence and function in ACs. Binary string systems were chosen for investigation because of the sequence-function analogy between binary strings and autocatalytic ribonucleic acids. However, despite the similari-

ties, there are several factors that are vastly different between binary string systems and organizations of pre-biotic molecules. How constructive interactions between objects can be is determined by the structure of artificial objects. Autocatalytic nucleic acids are 3-dimensional macromolecules consisting of a phosphate-sugar backbone, and different N-bases attached to the ribose sugar at the 1' position. The function of the autocatalytic nucleic acid is determined by the base sequence of the nucleic acid. However, if the nucleic acid is not folded correctly, the function of the resulting macromolecule will be altered. The folding process for each macromolecule is spontaneous and is simultaneously dependent on the nucleic acid sequence and composition of the surrounding environment.

In the AC that we chose to examine, the mapping of a string into an operator does not occur spontaneously, and the mapping of strings into operators is not dependent on the environment. The fact that mapping does not occur spontaneously, and must always be imposed on objects, is one factor that has managed to evade attempts at being modelled. The question remains: in artificial systems, how does one obtain a functional object from a sequence object, without imposing a mapping onto the sequence object? This is one issue that we were not able to properly address. In our systems, we used four mappings that were considered biologically standard, and several systems used random or selective folding, which allowed combinations of mappings to be used in a single system. Unfortunately, because mapping methods in ACs must be programmed, they are always deterministic, and any error or variance introduced into the mapping process must be programmed as well. The effect of this is that strings will always fold and replicate perfectly, and any chance of error that might be made by organic machinery *in vivo* must be programmed into the system *in silico*.

The effect of the environment on mapping is a factor that we did not examine in our systems, but, at least, it is possible to incorporate the environment as an effector of mapping, as opposed to having the mapping process be completely spontaneous. There are multiple ways of incorporating the environment into the mapping process. A common method is to use parity checking [9]. In this method the parity value of the environment could be used in conjunction with the parity value of the strings in question to decide on a folding type. Another method would be to use the ratio of the total number of zero- and one-bits in the population (the one-zero population ratio), and the one-zero ratios of the individual strings in question, as effectors of the mapping process. Using the environment as an effector of mapping on a macroscopic level would use the ratios obtained in a combinatorial fashion to determine the mapping to be used. If the environment were used as an effector of mapping on the microscopic level, the internal rules of matrix multiplication would be replaced with arbitrarily chosen logical operators, greatly changing the function of the matrix operator. However, changing the internal rules of matrix multiplication would change the type of system being investigated from a ‘matrix multiplication chemistry’ to something more in line with a ‘logical matrix chemistry’.

Another limitation with binary systems is the extent to which they can be used to examine self-production and self-organization. The immediate limitation of using such a simple system is the limited diversity of the species available. The maximum complexity and emergent behaviour that systems are capable of exhibiting depends on the number of different types of object in the system. That is to say, systems with more types of objects are capable of displaying more complex and emergent behaviour than systems with fewer types of objects. There were only 16 species in the systems we

examined, and one of them, the destructor, was completely excluded from all systems because of its pathogenic nature. Although it is not difficult to create binary systems with more species, such systems were not examined here. Two ways of increasing the complexity of small binary string systems will be discussed in section 5.2.

There are many examples of self-production in the systems we examined. However, there is no mystery surrounding the phenomena, as self-replication in binary systems has been previously described, and examining the matrix multiplication reactions of self-replications reveals the deterministic and mathematical nature of binary string replication. The extent of self-organization in binary systems is surprising, and even small systems produced competitive and cooperative organizations. The most complex organization consisted of four groups, two of which mutually existed while sharing no beneficial reactions. The simplest stable organization consisted only of two string species, which stabilised each other through passive and active replication reactions.

Another problem with ACs has always been, and probably will always be, the size of the populations in ACs, and the time it takes for simulations to complete. Increasing both population size, and simulation length, often exponentially increases the amount of time it takes for a simulation to complete. If system A consists of X strings, and system B consists of $Y \cdot X$ strings, and both simulations are run for Z generations, then system B will always go through Y times more iterations than system A, no matter how many generations pass.

5.2 Future research

In all of the systems that we investigated, the interactions between objects produced new objects. This is not the only method of operation, and, as detailed in Section 3.4, there are two other ways of producing strings and dealing with the change in population. It would be interesting to compare systems that produce new objects and remove random strings, with systems that essentially modify the string being operated on.

The second future objective would be to incorporate and examine the effect of the environment as a local effector. On a macroscopic level, the one-zero ratio of the population, and the one-zero ratio of the strings in question, could be simultaneously used to determine the appropriate folding. In an example of such a system, the mapping mechanism would use one of the two ratios obtained (population ratio and interacting string ratio) to determine whether the operator should fold canonically or topologically, and it would use the other ratio to determine whether the folding type should be transposed or normal. This could have long term effects on systems, because the choice of folding type will be continuously altered with the change in population, and systems may be stable up to a point where they reach a threshold for change, which could completely alter the dynamic of the system.

Another interesting phenomenon that arises, because of the nature of mathematics, is the inequality between the zero-bit and the one-bit during multiplication. Zero will always have the upper hand as a self-replicating bit during multiplication: anything multiplied by zero, becomes zero. A simple way of overcoming this bias is to change the internal function of the matrix multiplication, from multiplication to another function. Two appropriate mathematical logic operations that have no bias towards zeros

or ones are the 'exclusive or' (XOR) and 'equivalence' (AND) functions. The XOR function will output a zero value if the two input values are the same, and it will output a one value if the two input values are different. The AND function, in a sense, is exactly the opposite of the XOR function, as it requires both input values to be the same to output a one value, if the two input values differ then the output is a zero value.

A simple way to increase the complexity and emergent behaviour of the system would be to allow the formation of strings of variable length. Although this has been extensively studied, the addition of variable length strings, in combination with the previously mention future research prospects, would certainly yield interesting results and new types of organizations. A good starting point to limit the possible complexity of such systems would be to start by examining systems with strings of length four-bit and less.

Bibliography

- [1] Banzhaf, W. [1993] “Self-replicating sequences of binary numbers” *Computers & Mathematics with Applications* **26**, 1–8.
- [2] Banzhaf, W. [1993] “Self-replicating sequences of binary numbers. foundations I: General” *Biological Cybernetics* **69**, 269–274.
- [3] Banzhaf, W. [1993] “Self-replicating sequences of binary numbers. foundations II: Strings of length $N=4$ ” *Biological Cybernetics* **69**, 275–281.
- [4] Banzhaf, W. [1994] “Self-organization in a system of binary strings” in *Proceedings of Artificial Life IV* pp. 109–118.
- [5] Banzhaf, W. [1994] “Self-replicating sequences of binary numbers: The build-up of complexity” *Complex Systems* **8**, 215–225.
- [6] Banzhaf, W. [1995] “Self-organizing algorithms derived from RNA interactions” in *Evolution and biocomputation* vol. 899 pp. 69–102 Springer.
- [7] Banzhaf, W. [2004] “Artificial chemistries—towards constructive dynamical systems” *Solid State Phenomena* **97**, 43–50.

- [8] Banzhaf, W., Dittrich, P. and Eller, B. [1997] “Selforganization in a system of binary strings with topological interactions” *Physica D* **125**, 85–104.
- [9] Banzhaf, W., Dittrich, P. and Rauhe, H. [1996] “Emergent computation by catalytic reactions” *Nanotechnology* **7**, 307.
- [10] Barbieri, M. [2003] *The organic codes: an introduction to semantic biology* Cambridge university press.
- [11] Cariani, P. [1991] “Emergence and artificial life” *Artificial Life II* **10**, 775–798.
- [12] Centler, F., Kaleta, C., di Fenizio, P. and Dittrich, P. [2008] “Computing chemical organizations in biological networks” *Bioinformatics* **24**, 1611–1618.
- [13] Centler, F., Kaleta, C., di Fenizio, P. S. and Dittrich, P. [2010] “A parallel algorithm to compute chemical organizations in biological networks” *Bioinformatics* **26**, 1788–1789.
- [14] di Fenizio, P. S. and Banzhaf, W. [2000] “A less abstract artificial chemistry” *Artificial Life* **7**, 49–53.
- [15] di Fenizio, P. S., Dittrich, P., Banzhaf, W. and Ziegler, J. [2000] “Towards a theory of organizations” in *German Workshop on Artificial Life (GWAL 2000)*, in print, Bayreuth vol. 5 (7) p. 2000.
- [16] Dittrich, P. [1998] “Real evolution in artificial chemistries” *The Right Stuff: Appropriate Mathematics for Evolutionary and Developmental Biology* **6**, 27–31.

- [17] Dittrich, P. [1999] “Artificial chemistries” Tutorial held at European Conference on Artificial Life.
- [18] Dittrich, P. [2005] “Chemical computing” in *Unconventional programming paradigms* vol. 3566 pp. 19–32 Springer.
- [19] Dittrich, P. [2009] “Artificial chemistry” in *Encyclopedia of Complexity and Systems Science* pp. 326–344 Springer.
- [20] Dittrich, P. and Banzhaf, W. [1997] “A topological structure based on hashing-emergence of a spatial organization” in *Poster presented at the Fourth European Conference on Artificial Life , Brighton, UK*.
- [21] Dittrich, P. and Banzhaf, W. [1998] “Self-evolution in a constructive binary string system” *Artificial Life* **4**, 203–220.
- [22] Dittrich, P., Banzhaf, W., Rauhe, H. and Ziegler, J. [1997] “Macroscopic and microscopic computation in an artificial chemistry” in *Draft Proc. Second German Workshop on Artificial Life* pp. 19–22.
- [23] Dittrich, P. and di Fenizio, P. [2007] “Chemical organisation theory” *Bulletin of mathematical biology* **69**, 1199–1231.
- [24] Dittrich, P. and Ziegler, J. [1998] “Artificial chemistries” in *Third German Workshop on Artificial Life* p. 249.
- [25] Dittrich, P., Ziegler, J. and Banzhaf, W. [1998] “Mesoscopic analysis of self-evolution in an artificial chemistry” in *Artificial Life VI* pp. 95–103.
- [26] Dittrich, P., Ziegler, J. and Banzhaf, W. [2001] “Artificial chemistries—a review” *Artificial life* **7**, 225–275.

- [27] Eigen, M. [1971] “Selforganization of matter and the evolution of biological macromolecules” *Naturwissenschaften* **58**, 465–523.
- [28] Fontana, W. [1992] “Algorithmic chemistry” in *Artificial Life II* (Langton, C. G., Taylor, C., Farmer, J. D. and Rasmussen, S., eds.) pp. 159–209 Addison-Wesley, Redwood City, CA.
- [29] Fontana, W. and Buss, L. W. [1994] “The arrival of the fittest: Toward a theory of biological organization” *Bulletin of Mathematical Biology* **56**, 1–64.
- [30] Hofmeyr, J.-H. S. [2007] “The biochemical factory that autonomously fabricates itself: A systems-biological view of the living cell” in *Systems Biology: Philosophical Foundations* (Boogerd, F. C., Bruggeman, F., Hofmeyr, J.-H. S. and Westerhoff, H. V., eds.) chap. 10, pp. 217–242 Elsevier, Amsterdam.
- [31] Hofstadter, D. R. [1979] *Godel, Escher, Bach: An Eternal Golden Braid* New York: Basic Books, Inc.
- [32] Hutton, T. [2007] “The organic builder: A public experiment in artificial chemistries and self-replication” *Artificial Life* **15**, 21–28.
- [33] Kaplan, D. M. and White, C. G. [2003] “Hands-on electronics” *Hands-On Electronics, by Daniel M. Kaplan and Christopher G. White, pp. 226. Cambridge University Press, May 2003* **1**, 65–67.
- [34] Kvasnicka, V. and Pospichal, J. [2001] “Autoreplicators and hypercycles in typogenetics” *Journal of Molecular Structure: THEOCHEM* **547**, 119–138.

- [35] Lazcano, A. and Bada, J. L. [2003] “The 1953 Stanley L. Miller experiment: fifty years of prebiotic organic chemistry” *Orig. Life Evol. Biosph.* **33**, 235–242.
- [36] Matsumaru, N. and Centler, F. [2004] “Self-adaptive scouting autonomous experimentation for systems biology” in *Applications of Evolutionary Computing* pp. 52–62 Springer.
- [37] Matsumaru, N., Centler, F., di Fenizio, P. S. and Dittrich, P. [2007] “Chemical organization theory as a theoretical base for chemical computing.” *International Journal of Unconventional Computing* **3**.
- [38] Matsumaru, N., di Fenizio, P. S., Centler, F. and Dittrich, P. [2006] “On the evolution of chemical organizations” in *Proc. of the 7th German Workshop of Artificial Life* pp. 135–146.
- [39] Morris, H. C. [1987] “Typogenetics: A logic for artificial life.” in *Artificial Life* pp. 369–396.
- [40] Morris, H. C. [1989] *Typogenetics: a logic of artificial propagating entities* Ph.D. thesis University of British Columbia.
- [41] Ono, N. and Ikegami, T. [1999] “Model of self-replicating cell capable of self-maintenance” in *Advances in artificial life* pp. 399–406 Springer.
- [42] Ono, N. and Ikegami, T. [2000] “Self-maintenance and self-reproduction in an abstract cell model” *Journal of Theoretical Biology* **206**, 243–253.
- [43] Rasmussen, S., Knudsen, C., Feldberg, R. and Hindsholm, M. [1990] “The coreworld: Emergence and evolution of cooperative structures

- in a computational chemistry” *Physica D: Nonlinear Phenomena* **42**, 111–134.
- [44] Rosen, R. [1991] *Life Itself: a comprehensive inquiry into the nature, origin, and fabrication of life* Columbia University Press, New York.
- [45] Turing, A. M. [1952] “The chemical basis of morphogenesis” *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences* **237**, 37–72.
- [46] Varela, F. G., Maturana, H. R. and Uribe, R. [1974] “Autopoiesis: the organization of living systems, its characterization and a model” *Biosystems* **5**, 187–196.
- [47] Vareto, L. [1993] “Typogenetics: An artificial genetic system” *Journal of Theoretical Biology* **160**, 185–205.
- [48] Vareto, L. [1998] “Studying artificial life with a molecular automaton” *Journal of Theoretical Biology* **193**, 257–285.
- [49] von Neumann, J. and Burks, A. W. [1966] *Theory of self-reproducing automata* University of Illinois Press, Urbana, Illinois.
- [50] Wills, P. R. [2011] “Life requires genetic representation and vice versa—consequences for alife” *Advances in Artificial Life: ECAL11* pp. 866–873.

Appendix

Program listing

```
import pysces, random as rdm, numpy as np, scipy,
        matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator

class Replicator(object):

# STEP 1: Generate a population of random four-bit binary strings.
# The .gen command generates a population of size 'pop' and appends
# them to the list 'soup'.
    def gen(self):
        for i in xrange(pop):
            x=rdm.randint(0,15)
            soup.append(np.binary_repr(x, width=4))

# STEP 1(ALT): Generate a homogeneous population of binary strings.
# The .seed command will generate a homogeneous population of string
# 'x', with a size of 'pop', and append them to the list 'soup'.
    def seed(self,x):
        for i in xrange(pop):
            soup.append(np.binary_repr(x, width=4))

# MAPPING FUNCTIONS
# This section contains the mechanisms behind each folding type.
# If the command is not given an operator then both the operator (op)
# and string (st) will be randomly selected.
# The .canonical command returns the product string of 'op' and 'st'.
    def canonical(self,op,st):
```

```

    if op=='*':
        samp=rdm.sample(soup,2)
        op=samp[0]
        st=samp[1]
    a=int(op[3])*int(st[3])+int(op[2])*int(st[2])
    if a>1:
        a=1
    b=int(op[1])*int(st[3])+int(op[0])*int(st[2])
    if b>1:
        b=1
    c=int(op[3])*int(st[1])+int(op[2])*int(st[0])
    if c>1:
        c=1
    d=int(op[1])*int(st[1])+int(op[0])*int(st[0])
    if d>1:
        d=1
    ns=str(d)+str(c)+str(b)+str(a)
    return ns
# The .transpose_canonical command returns the product string of
# 'op' and 'st'.
def transpose_canonical(self,op,st):
    if op=='*':
        samp=rdm.sample(soup,2)
        op=samp[0]
        st=samp[1]
    a=int(op[3])*int(st[3])+int(op[1])*int(st[2])
    if a>1:
        a=1
    b=int(op[2])*int(st[3])+int(op[0])*int(st[2])
    if b>1:
        b=1
    c=int(op[3])*int(st[1])+int(op[1])*int(st[0])
    if c>1:
        c=1
    d=int(op[2])*int(st[1])+int(op[0])*int(st[0])
    if d>1:
        d=1
    ns=str(d)+str(c)+str(b)+str(a)
    return ns
# The .topological command returns the product string of
# 'op' and 'st'.

```

```

def topological(self, op, st):
    if op==' ':
        samp=rdm.sample(soup, 2)
        op=samp[0]
        st=samp[1]
    a=int(op[3])*int(st[3])+int(op[2])*int(st[2])
    if a>1:
        a=1
    b=int(op[0])*int(st[3])+int(op[1])*int(st[2])
    if b>1:
        b=1
    c=int(op[3])*int(st[1])+int(op[2])*int(st[0])
    if c>1:
        c=1
    d=int(op[0])*int(st[1])+int(op[1])*int(st[0])
    if d>1:
        d=1
    ns=str(d)+str(c)+str(b)+str(a)
    return ns
# The .transpose_topological command returns the product string of
# 'op' and 'st'.
def transpose_topological(self, op, st):
    if op==' ':
        samp=rdm.sample(soup, 2)
        op=samp[0]
        st=samp[1]
    a=int(op[3])*int(st[3])+int(op[0])*int(st[2])
    if a>1:
        a=1
    b=int(op[2])*int(st[3])+int(op[1])*int(st[2])
    if b>1:
        b=1
    c=int(op[3])*int(st[1])+int(op[0])*int(st[0])
    if c>1:
        c=1
    d=int(op[2])*int(st[1])+int(op[1])*int(st[0])
    if d>1:
        d=1
    ns=str(d)+str(c)+str(b)+str(a)
    return ns

```

```

# The .operate command is used to initiate and maintain the algorithm
# that drives the interactions in the population. The command
# handles 'x', which is the number of iterations the system must
# perform, and 'fold', which determines the folding type applied to
# the string to map it into an operator. 'L', 'c' and 'a' are used
# ensure the system only saves 1000 data points per simulation.
# 'Fcount' records the number of times each folding type was used
# during the simulation. 'Fcount', 'fold' and 'a' are all passed
# into the .grow_op command, which performs the operations.
# Substitution is also toggled here for systems of fixed size

# STEP 7: Select a random string for evaluation for substitution.
# The .substitute command evaluates a string and replaces it with a
# random member of the population if it fails the selection process.
    def operate(self,x,fold):
        L=(x*pop)/1000
        c=0
        Fcount=[0,0,0,0,0]
        print str(x)+' Generations'
        for i in xrange(x*pop):
            a=0
            if c==0:
                a=1
            c=c+1
            if c==L:
                c=0
            self.grow_op(fold,a,Fcount)
            self.substitute()
            print 'Loading: '+str(i*100./(x*pop))+ '%'
        self.show(fold)
        print Fcount

# The .grow_op command contains STEPS 2-5
# STEP 2: Select a random string and fold it into an operator.
# Two random strings, 'op' and 'st', are selected from the population
# The 'fold' value decides how the operator will be mapped.
# There are six different folding types that can be selected:
# canonical='cf', transposed canonical='tc', topological='tp',
# transposed topological='tt', random='ran', selective='ns',
# STEP 3: Apply the operator to a second randomly selected string,
#         generating a new string.

```

```
# The strings 'op' and 'st' are passed to one of the folding types,
# which will return the newly generated string.
# STEP 4: Release the new string into the soup if it complies with
#         the selection criteria.
# Once the new string has been generated it is evaluated by the
# selection criteria and appended to the population if it complies.
# STEP 5: If the operation does not comply with the reaction rules,
#         restart the current reaction from STEP 2.
# If the string produced in STEP 3 does not comply with the
# selection criteria, then process is repeated from STEP 2.
# STEP 6: Remove a random string to compensate for the addition.
# If STEP 4 was successful then a random string must be removed from
# the population to compensate for the additional string.
def grow_op(self, fold, a, Fcount):
    sample=rdm.sample(soup, 2)
    op=sample[0]
    st=sample[1]
    ns=''
    if fold=='cf':
        ns=self.canonical(op, st)
        Fcount[0]=Fcount[0]+1
    if fold=='tc':
        ns=self.transpose_canonical(op, st)
        Fcount[1]=Fcount[1]+1
    if fold=='tp':
        ns=self.topological(op, st)
        Fcount[2]=Fcount[2]+1
    if fold=='tt':
        ns=self.transpose_topological(op, st)
        Fcount[3]=Fcount[3]+1
    if fold=='ran':
        x=rdm.randint(0, 3)
        if x==0:
            ns=self.canonical(op, st)
            Fcount[0]=Fcount[0]+1
        if x==1:
            ns=self.transpose_canonical(op, st)
            Fcount[1]=Fcount[1]+1
        if x==2:
            ns=self.topological(op, st)
            Fcount[2]=Fcount[2]+1
```

```

    if x==3:
        ns=self.transpose_topological(op, st)
        Fcount[3]=Fcount[3]+1
i2=int(op,2)
if fold=='ns':
    tot=complex[i2,0]+complex[i2,1]+
        complex[i2,2]+complex[i2,3]
    hit=rdm.randint(1,tot)
    if hit>=1 and hit<=complex[i2,0]:
        ns=self.canonical(op, st)
        Fcount[0]=Fcount[0]+1
    if hit>=(complex[i2,0]+1) and hit<=(complex[i2,0]+
        complex[i2,1]):
        ns=self.transpose_canonical(op, st)
        Fcount[1]=Fcount[1]+1
    if hit>=(complex[i2,0]+complex[i2,1]+1) and
    hit<=(complex[i2,0]+complex[i2,1]+complex[i2,2]):
        ns=self.topological(op, st)
        Fcount[2]=Fcount[2]+1
    if hit>=(complex[i2,0]+complex[i2,1]+complex[i2,2]+1)
    and hit<=tot:
        ns=self.transpose_topological(op, st)
        Fcount[3]=Fcount[3]+1
    if op==ns:
        if hit>=1 and hit<=complex[i2,0]:
            complex[i2,0]=complex[i2,0]+1
        if hit>=(complex[i2,0]+1) and hit<=(complex[i2,0]+
            complex[i2,1]):
            complex[i2,1]=complex[i2,1]+1
        if hit>=(complex[i2,0]+complex[i2,1]+1) and
        hit<=(complex[i2,0]+complex[i2,1]+complex[i2,2]):
            complex[i2,2]=complex[i2,2]+1
        if hit>=(complex[i2,0]+complex[i2,1]+complex[i2,2]+1)
        and hit<=tot:
            complex[i2,3]=complex[i2,3]+1
if ns!='0'*4: # for making s15 elastic include:
    s=rdm.randint(0, len(soup)-1) # and ns!='1'*4
    soup.append(ns)
    soup.pop(s)
if a==1:
    data.append(self.percent())

```

```

    if ns=='0'*4:                # for making s15 elastic include:
        self.grow_op(fold , a, Fcount)          # or ns=='1'*4
        Fcount[4]=Fcount[4]+1

# In addition to being able to remove random strings, the .delete
# command is also able to remove the given string or operator.
def delete(self,x,ch,op,st):
    if ch=='ran':
        s=rdm.randint(0,len(soup)-1)
        soup.pop(s)
    if ch=='str':
        pos=soup.index(st)
        soup.pop(pos)
    if ch=='op':
        pos=soup.index(op)
        soup.pop(pos)

# The .substitute command evaluates a string and replaces it with a
# random member of the population if it fails the selection process.
def substitute(self):
    samp=rdm.sample(soup,1)
    x=0
    for l in xrange(4):
        x=x+int(samp[0][l])
    prob=(x/4.)*1 # the threshold probability value
    c=rdm.random() # is toggled here, after the '**'
    if prob>c:
        add=rdm.randint(0,pop-1)
        pos=soup.index(samp[0])
        soup.append(soup[add])
        soup.pop(pos)

# The .grow command is only used when it is desired that the
# population must continue growing with each iteration, as opposed
# to the system remaining stable.
def grow(self,x,T):
    for i in xrange(x):
        self.grow_op(1,T)
        self.substitute()
        print 'Loading: '+str(i*100./x)+'%'
    self.show()

```

```
# The .show show command is used to display the current data set as
# an image.
def show(self,z):
    x=[]
    if z=='cf':
        z='canonical'
    if z=='tc':
        z='transpose canonical'
    if z=='tp':
        z='topological'
    if z=='tt':
        z='transpose topological'
    if z=='ran':
        z='random'
    if z=='ns':
        z='selective'
    plt.clf()
    plt.plot(data,linewidth=1)
    for i in range(15):
        x.append('s'+str(i+1)+' : '+str(data[len(data)-1][i])+'%')
    plt.title('4-bit '+z+' folding')
    plt.ylabel('String concentration(%)')
    plt.xlabel('Data points')
    self.check()

# The .save command is used to save the current image and data set.
def save(self,y,z):
    x='4bitG'+str(y)+'P'+str(len(soup)/1000)+'ks'+str(S)+str(z)
    plt.savefig(x+'.pdf')
    np.save(x,data)

# The .load command is used to load a previously saved data set.
def load(self,data,fold):
    plt.clf()
    plt.plot(data)
    if fold=='':
        fold='canonical'
    if fold=='tc':
        fold='transpose canonical'
    if fold=='tp':
```

```

        fold='topological'
    if fold=='tt':
        fold='transpose topological'
    if fold=='ran':
        fold='random'
    if fold=='ns':
        fold='selective'
    plt.title('4-bit ' + fold + ' folding')
    plt.ylabel('String concentration(%)' )
    plt.xlabel('Data points')

# The .check command is used to confirm the population size.
def check(self):
    print 'Soup length: ' + str(len(soup))
    r.species()

# The .species command is used to display all species concentrations.
def species(self):
    conc=np.zeros((16))
    for i in xrange(0,16):
        conc[i]=soup.count(np.binary_repr(i, width=4))
        conc[i]=conc[i]/int(len(soup))*100.
        print str(np.binary_repr(i, width=4)) + ': ' + str(conc[i]) + '%'

# The .percent command is used to generate savable data.
def percent(self):
    spec=np.zeros((15))
    for i in xrange(15):
        spec[i]=soup.count(np.binary_repr(i+1, width=4))
        spec[i]=spec[i]/int(len(soup))*100.
    return spec

# The .reaction_table command is used to display the desired
# reaction table.
def reaction_table(self, fold):
    nsg, rep, srep=0,0,0
    table=np.zeros((16,16))
    histo=np.zeros((16))
    hfreq=np.zeros((16))
    title=''
    if fold=='':

```

```

        title='Canonical folding'
plt.clf()
plt.axis([-1,16,-1,16])
for x in xrange(16):
    for y in xrange(16):
        mst=np.zeros((4))
        mop=np.zeros((4))
        st=str(np.binary_repr(x,width=4))
        op=str(np.binary_repr(y,width=4))
        ns=self.canonical(op,st)
        if fold=='tc':
            ns=self.transpose_canonical(op,st)
            title='Transpose Canonical folding'
        if fold=='tp':
            ns=self.topological(op,st)
            title='Topological folding'
        if fold=='tt':
            ns=self.transpose_topological(op,st)
            title='Transpose Topological folding'
        col=ns
        s3=int(col,2)
        table[y][x]=s3
        histo[s3]=histo[s3]+1.
        if col!=st and col!=op:
            col='blue'
            nsg=nsg+1
        if col==st and col==op:
            col='red'
            srep=srep+1
        if col==st or col==op:
            col='green'
            rep=rep+1
        plt.text(x-0.1,y-0.1,int(s3),color=col,fontsize=15)
plt.title(title)
plt.ylabel('Operator')
plt.xlabel('String')
ax=plt.subplot(111)
ax.xaxis.set_major_locator(MultipleLocator(1))
ax.yaxis.set_major_locator(MultipleLocator(1))
plt.savefig(title+'.pdf')
hfsum=0

```

```

histsum=0
for i in xrange(16):
    hfreq[i]=histo[i]/256.*100.
    hfsum=hfsum+hfreq[i]
    histsum=histsum+histo[i]
return table ,srep , rep , nsg , histo , histsum , hfreq , hfsum

# The .dependence command will return the dependency values for the
# given list , 'l', of strings.
def dependence(self,l):
    dep=np.zeros((16,16))
    sum=np.zeros((16))
    for s in range(2):
        for ts in range(4):
            if ts==0:tab=self.reaction_table('cf')
            if ts==1:tab=self.reaction_table('tc')
            if ts==2:tab=self.reaction_table('tp')
            if ts==3:tab=self.reaction_table('tt')
            for y in range(16):
                for l1 in range(len(l)):
                    if y==int(l[l1]):
                        for x in range(16):
                            for l2 in range(len(l)):
                                if x==int(l[l2]):
                                    if s==0:
                                        sum[int(tab[0][y][x])]=
                                        sum[int(tab[0][y][x])+1]
                                    if s==1:
                                        dep[int(tab[0][y][x])[y]=
                                        dep[int(tab[0][y][x])[y]+
                                        1./sum[int(tab[0][y][x])]*50.
                                        dep[int(tab[0][y][x])[x]=
                                        dep[int(tab[0][y][x])[x]+
                                        1./sum[int(tab[0][y][x])]*50.

plt.close()
return dep

# This is where important system parameters are set.
# The default folding type 'fold,' is initialised as canonical
# folding. The 'pop' parameter is the size of the population to be
# initialised. The list , 'soup', is the holder for the string

```

Program listing

```
# population. The list, 'data', will hold arrays that contain the
# string species for each data point saved and displayed. The
# 'complex' array contains the memory values for strings when systems
# utilize selective folding. The 'S' integer is toggled for
# homogeneous systems that are initiated with only one string type.
if __name__ is '__main__':
    fold='cf'
    pop=10000
    soup=[]
    data=[]
    complex=np.zeros((16,4))
    complex[:,:] = pop/40
    S=1

    r = Replicator()
    r.gen()
    r.seed(S)
    r.check()
```
