J.A. Reggia, H.-H. Chou, and J.D. Lohn. ``Cellular Automata Models of Self-replicating Systems," in Advances in Computers, M. Zelkowitz (ed), vol. 47, Academic Press, New York, 1998, pp.141-183.

Cellular Automata Models of Self-Replicating Systems

James A. Reggia*
Dept. of Computer Science & Institute for Advanced Computer Studies
A. V. Williams Bldg., University of Maryland, College Park, MD 20742 USA
reggia@cs.umd.edu

Hui-Hsien Chou
The Institute for Genomic Research, 9712 Medical Center Drive
Rockville, MD 20850 USA
hhchou@tigr.org

Jason D. Lohn
Caelum Research Corporation
NASA Ames Research Center, MS 269-1, Moffett Field, CA 94035 USA
jlohn@ptolemy.arc.nasa.gov

Abstract: Since von Neumann's seminal work around 1950, computer scientists and others have studied the algorithms needed to support self-replicating systems. Much of this work has focused on abstract logical machines (automata) embedded in two-dimensional cellular spaces. This research has been motivated by the desire to understand the basic information processing principles underlying self-replication, the potential long term applications of programmable self-replicating machines, and the possibility of gaining insight into biological replication and the origins of life. Here we briefly summarize the historical development of work on artificial self-replicating structures in cellular spaces, and then describe some recent advances in this area. Past research is viewed as taking three main directions: early complex universal computer-constructors modeled after Turing machines, qualitatively simpler self-replicating loops, and efforts to view self-replication as an emergent phenomenon. We discuss our own recent studies showing that self-replicating structures can emerge from non-replicating components and that genetic algorithms can be applied to automatically program simple but arbitrary structures to replicate. We also describe recent work in which self-replicating structures are successfully programmed to do useful problem solving as they replicate. We conclude by identifying some implications and important research directions for the future.

*To whom correspondence should be sent.

CONTENTS

1. WHY STUDY SELF-REPLICATING SYSTEMS?

Self-replicating systems are systems that have the ability to produce copies of themselves. Biological organisms are, of course, the most familiar examples of such systems. However, around 1950 mathematicians and computer scientists began studying artificial self-replicating systems in order to gain a deeper understanding of complex systems and the fundamental information-processing principles involved in self-replication [5, 70]. The initial models that were developed consisted of abstract logical machines, or automata, embedded in cellular spaces [2, 11, 33, 57]. In addition to work on cellular automata, other computational models, such as those based on more traditional programming concepts [56], continue to be the subject of research. Mechanical and biochemical models have also been constructed and studied [26, 46, 49].

Much of this work on artificial self-replicating systems has been motivated by the desire to understand the fundamental information processing principles and algorithms involved in self-replication, independent of how they might be physically realized. A better theoretical understanding of these principles could be useful in a number of ways from a computational/engineering perspective. For example, it has been proposed that:

Self-replicating programs undergoing artificial selection could facilitate the difficult task of programming massively parallel computers. Experiments performed on sequential computers have shown that such programs can optimize their algorithms more than fivefold in a few hours of time [56].

Understanding self-replication processes could shed light on computer viruses and may contribute to their detection and the creation of biologically-inspired "immune systems" [30].

Self-replicating devices could play a key role in atomic-scale manufacturing or "nanotechnology" [13]. Researchers in this area have already gained insight from early work on self-replicating systems [41].

Year	*Model Type*	*Dim.*	*Rot. Symmetry*	*States per Cell*	*Neighborhood size(s)*	*Structure size(s)*[a]	*Capabilities*[b]	*Refs.*
1951	CA	2D	weak	29	5	$> 10^4$	o	[70, 51]
1965	CA	2D	strong	8	5	$> 10^4$	o	[11]
1966	CT-Mach.	2D	weak	$\approx 10^{100}$	5	$\approx 10^2$	o	[2]
1973	CA	2D	strong	8	5	$> 10^4$	s	[68]
1976	α-Univ.	1D	strong	5	-[c]	(60)[d]	s	[25]
1984	CA	2D	strong	8	5	86	s	[33]
1989	CA	2D	strong	6	5	12	s	[6]
1993	CA	2D	both	6,8	5,9	5–48	s	[57]
1995	EA	2D	weak	9,13	5	2,3	s	[36]
1995	CA	2D	strong	6	9	52	o	[65]
1995	non-uni. CA	2D	strong	2	9	5	s	[61]
1996	CA/W-Mach.	2D	strong	63	5	127	o	[50]
1997	CA	2D	weak	192	9	4 or more	s	[8]

[a]Some systems were never implemented, thus certain values are approximations.
[b]s=self-replication, o=other capabilities in addition to self-replication.
[c]No fixed neighborhood size.
[d]Theorized.

Table 1: Examples of research involving self-replicating structures in cellular space models.

Self-replicating systems may have an important future role in planetary exploration [16] and in creating robust electronic hardware [39, 40].

Developing an understanding of the principles of self-replication is also of interest in a broader scientific context. For example, understanding these principles may advance our knowledge of the biomolecular mechanisms of reproduction, clarifying conditions that any self-replicating system must satisfy and providing alternative explanations for empirically observed phenomena. Self-replicating systems have thus become a major area of research activity in the field of artificial life [34, 35]. Work in the area of self-replicating systems could shed light on those contemporary theories of the origins of life that postulate a prebiotic period of molecular replication before the emergence of living cells [46, 47, 52].

2. EARLY SELF-REPLICATING STRUCTURES

Table 1 lists several examples of past cellular automata studies of self-replicating structures. While the earliest work on artificial self-replicating structures/machines sometimes used mechanical devices [29, 49], subsequent work has been based largely upon computational modelling, especially with cellular automata. Thus, in the rest of this article, we will focus on self-replicating structures implemented within the framework of cellular automata. Work on such models can be viewed as primarily taking three approaches. First, early self-replicating structures (1960's and 1970's) were large, complex universal systems modeled after Turing machines. Although these early models were so large and complex that they have never actually been fully implemented, they provided the first demonstration that artificial self-replicating structures could in principle be devised and stimulated substantial theoretical work. These early models are discussed in Sections 2.2 and 2.3. A second generation of self-replicating structures, studied since the mid 1980's, were designed to be qualitatively simpler than their predecessors. This was done by relaxing the criteria that self-replicants must also be capable of universal computation and construction. These models, characterized as self-replicating loops, are discussed in Section 3. More recently, we have taken a third approach in our own work that focuses on self-replication as an emergent property rather than, as in the past, being based solely on manually designed replicants. This work has shown that self-replicating structures can emerge from initially random states, and that rules to control self-replication can be discovered using artificial evolution methods (genetic algorithms). It has also been established that self-replicating structures can be used to solve problems while they replicate. These recent developments are summarized in Sections 4 and 5. Finally, Section 6 speculates about some of the implications of this work and offers suggestions for future work.

2.1 Cellular Automata Framework

Since the models of self-replication described below are implemented in a cellular automata framework, we briefly describe this framework here. *Cellular automata* can be characterized as an array of identical processing units called *cells* that are arranged and interconnected throughout space in a regular manner. Figure 1 shows a typical example of a small two-dimensional space tesselated into squares where each square represents one cell. Each cell represents the same abstract finite state *automaton* (computer), which typically can be in any of two or more possible *states*. These internal states are usually represented by letters, digits,

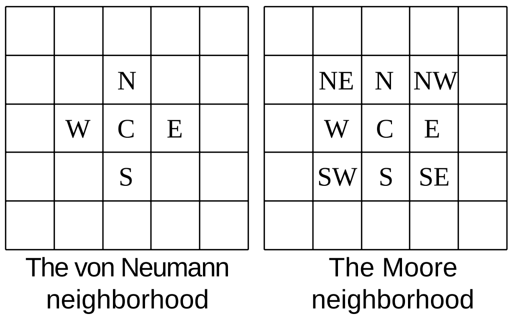


Figure 1: The von Neumann neighborhood (5-neighborhood) and Moore neighborhood (9-neighborhood) of a central cell labeled C; here N = north, E = east, S = south, and W = west. The labels here, unlike in all subsequent illustrations, do not represent cell states.

or other non-numeric characters. A special state, called the quiescent or inactive state, is generally represented by an empty cell in pictures, or as a period in text. All other cells are said to be active.

At each tick of simulated time, each cell simultaneously changes state as a function of its own current state and the state of its immediately neighboring cells. Which cells are considered to be immediate neighbors varies from model to model. With the 5-neighborhood or von Neumann neighborhood, each cell (such as the one marked C for "central" in the left half of Figure 1) is considered to have five immediate neighbors: north, east, south, west and itself. In other words, a cell makes a decision about its new state based on its four adjacent cells plus its own state. During processing, various *structures* (configurations) arise. A structure is a fixed/moving persistent pattern of multiple contiguous activated cells. For example, Figure 2 illustrates a structure called a "glider" from the well-known cellular automata model called the Game of Life [4, 17]. The Game of Life model uses the 9-neighborhood or "Moore neighborhood" (Fig. 1, right). Cells in the Game of Life have only two possible states, dead (quiescent, indicated by empty cells) or alive (indicated by 1's). Thus, in Fig. 2 at the start (upper left), exactly five cells are alive, and these form a structure called a glider. At each instant of time t, each cell follows a simple set of rules forming the *transition function*, based on the number N of its eight neighbors that are alive (in state 1):

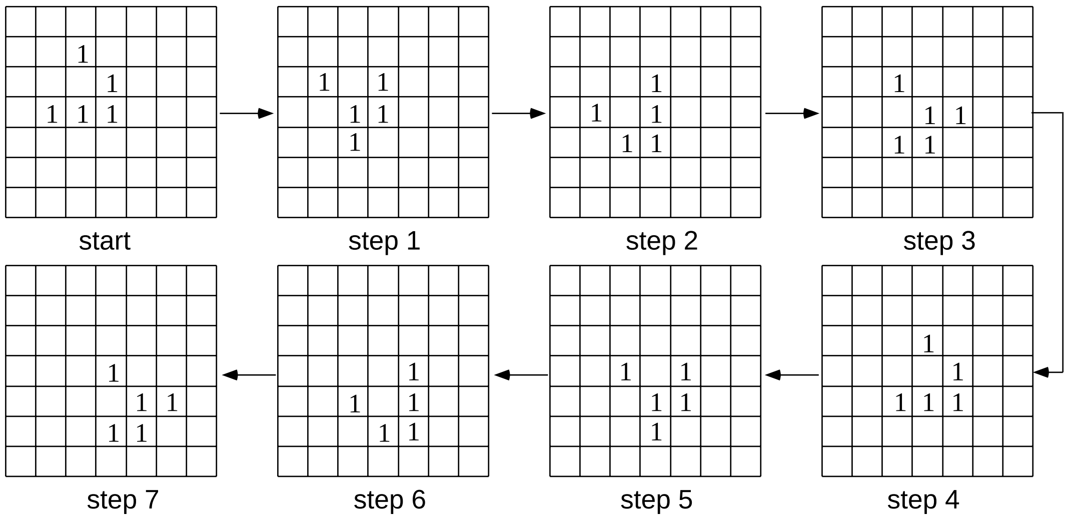


Figure 2: Successive applications of the Game of Life rules given in the text to a small initial cellular automata structure called a *glider*. The glider gradually moves to the lower right as each cell follows the same set of instructions based solely on local information. For example, at iteration 4 (lower right) it can be seen that the initial configuration has reappeared but shifted one space down and to the right.

```
If quiescent at time t and N=3,
     then at time t+1 change state to 1 ("birth").
If alive at time t and either N < 2 or N > 3,
     then at time t+1 become quiescent ("death").
Otherwise, stay in the same state at time t+1 as at time t.
```

Following these rules, the glider structure, the pattern of 1's in Figure 2, goes through a sequence of steps that shift it one unit diagonally every four units of time. Again, we follow the convention here and in the following figures of showing quiescent cells as being empty or blank. We refer to each iteration of the model in which all cells simultaneously change state as one step or one unit of time.

As with the Game of Life, with any cellular automata model each cell's state transitions are governed by a set of rules forming the transition function. Each single rule is simple and based solely on locally-available information. The "locality" of computation, that is, the fact that each cell can change its state based only on the state of its neighbors (including its own current state), is a fundamental aspect of cellular automata computation. In spite of such localized information processing, experience has shown that the complete set of rules forming a transition function, through their application by all of the cells in the model simultaneously and repetitively over time, can produce very rich and at times striking behavior. For this reason, cellular automata are being increasingly used as models in physics, chemistry, biology, and other scientific fields. The critical point here is that, since all computations are strictly local operations, any fixed/moving/replicating structures that occur represent *emergent behavior* of the model. The reader interested in further details of cellular automata models in general is referred to the many collections and reviews on this topic [12, 14, 21, 53, 72, 73].

2.2 von Neumann's Universal Computer-Constructor

The mathematician John von Neumann first used cellular automata to study the logical organization of self-replicating structures [70]. In his and most subsequent work, two-dimensional cellular automata spaces are used, and cells can be in one of several possible states. At any moment most cells are quiescent or inactive; those cells that are active are said to be *components*. A self-replicating structure is represented as a configuration of contiguous active cells, each of which represents a component of a replicating machine. Put otherwise, there are actually *two* levels at which one can talk about "machines" in the models we consider below:

Cells: Each cell forming the cellular automaton space is a finite state machine. For simplicity, in the rest of this review this fact will largely be kept implicit. We will instead emphasize the view that a cell represents a local region of space, a quiescent cell represents empty space, an active cell represents a region containing a component of a structure, and the transition function (rules or program) followed by a cell represents the "underlying physics" of the space.

Structures: A set of contiguous active cells or components (such as the "glider" in Figure 2) can also be viewed as an abstract machine. Such a structure, spanning several cells of the cellular space and considered as a whole, is a machine at a higher level of abstraction. When we refer to self-replicating structures or "machines" in the following, we will be referring to this higher level of abstraction.

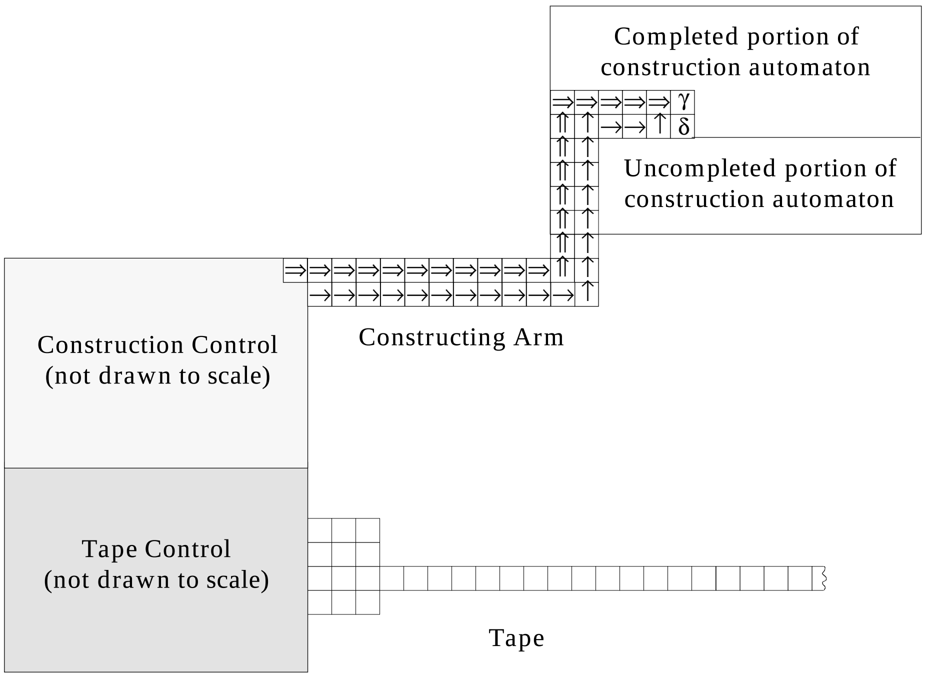

Completed portion of construction automaton
γ
δ
Uncompleted portion of construction automaton
Construction Control (not drawn to scale)
Constructing Arm
Tape Control (not drawn to scale)
Tape

Figure 3: Schematic diagram of von Neumann's self-replicating structure. The actual structure occupies at least tens of thousands of cells. The initial structure consists of construction and tape controls as shown shaded on the left. The tape contains a description of the initial structure to be replicated, the actual work being done by a constructing arm. A partially completed copy of the original structure is shown in the nearby cellular space (upper right). During the replication process, instructions on the tape cause the construction control to send signals up the constructing arm. These signals cause the arm to move as it sequentially deposits components in the replicant. (Figure taken from *Essays on Cellular Automata*, A. Burks (ed.), copyright 1970 by the Board of Trustees of the University of Illinois, used with permission of the University of Illinois Press).

Since at each instance of simulated time, each cell determines its next state as a function of only its current state and the state of immediate neighbor cells, any self-replicating structures observed in the models we consider must be an emergent behavior arising from strictly local interactions. Based solely on these concurrent local interactions an initially-specified self-replicating structure goes through a sequence of steps to construct a duplicate copy of itself (the replica being displaced and perhaps rotated relative to the original).

Von Neumann's original self-replicating structure is a complex universal computer-constructor embedded in a large, two-dimensional cellular automata space that consists of 29-state cells. It is based on the 5-neighborhood, and is literally a simulated digital computer

(Turing Machine) that useds a "construction arm" in a step-by-step fashion to construct a copy of itself from instructions on a "tape". In Fig. 3, the initial state of this structure is shown on the left (shaded) with its attached tape and its construction arm extended out to the upper right where a replicant is in the process of being constructed. The initial machine is said to be a *universal constructor* in that it can construct a copy of any structure properly specified on its tape [5]. It can also copy its input tape and attach it to the new structure. Self-replication can thus occur if the original machine is given a tape with a description of its own structure.

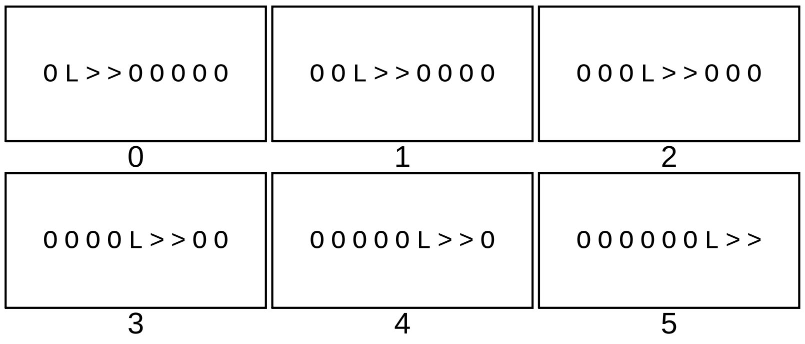


Figure 4: Signal sequence flow over a data path. Each box is a snapshot of the same region in the cellular automata space during successive times (iterations). Numbers below each box denote the times. The rules forming the transition function are such that signal > is followed by either a signal > or by the signal L. Signal L always changes to O, the latter changing to > if pointed at by >. The net effect is that the signal sequence L>> progressively moves to the right at a rate of one cell per unit time.

One of the important concepts introduced in von Neumann's universal computer-constructor is that of a *data path* over which signals can flow. The construction arm in Fig. 3 provides an example. Without describing the details of von Neumann's specific design, the basic idea of a data path and signal flow is illustrated in Fig. 4. The data path fragment shown in the figure consists of a row of cells in a state labeled by the letter O. Three signals (>>L) are embedded in the data path at $t = 0$, and at each iteration (tick of the clock) move one cell to the right. The transition rules obeyed by each and every individual cell that produce such behavior can be summarized as:

```
If in state 0 at time t and pointed at by >,
    then change to state > at time t+1.
If in state > at time t and a neighbor cell is in state L,
    then change to state L at time t+1.
If in state L at time t,
    then change to state 0 at time t+1.
Otherwise do not change state.
```

Data paths similar to that illustrated here serve as the "wires" over which information is transmitted, both internally in the universal computer-constructor and via the construction arm (Fig. 3).

Von Neumann's work provided an early demonstration that an artificial information-carrying system capable of self-replication was theoretically possible. It established, within the cellular automata framework, a logical organization that is sufficient for self-replication. The detailed design of von Neumann's original universal computer-constructor can be found in [70] and is clearly summarized in [5].

2.3 The Drive to Simplification

While the work by von Neumann established that artificial self-replication is possible, it left open the question of the minimal logical organization necessary for self-replication [5, 70]. Subsequent analysis led to several other results: it showed that some simplification of von Neumann's configuration was possible by redesigning specific components [66] or by increasing cell state complexity [2], demonstrated that sexual reproduction could be simulated [68], generalized von Neumann's basic result to other configurations and higher-dimensional cellular spaces [45], established theoretical upper bounds on how rapid a population of self-replicating configurations could grow [43], and examined several fundamental issues [3, 7, 22, 54, 60] that continue to generate theoretical interest today [28, 63].

Most influential among this early work has been Codd's demonstration that if the components or cell states meet certain symmetry requirements, then von Neumann's configuration could be done in a simpler fashion using cells having only eight states rather than the 29 used originally [11]. Codd argued that using components that were symmetrical led to a simpler model, and made modifications to von Neumann's design based on this and considerations about how brain cells transmit information. He also implemented and tested several parts of the replicating structure that he designed. His universal computer-constructor was simpler but otherwise similar in spirit to that of von Neumann.

Another approach taken to reducing the complexity of von Neumann's design focused on using more complex components [2]. The resulting 2-D cellular space model was referred to as *Constructing Turing Machines*, or *CT-machines* [66]. Each cell in this space contains a finite-state automata that executes short 22-instruction programs. The instructions consist of actions such as *weld* and *move*, and internal control constructs such as *if* and *goto*. Self-replication occurs when individual CT-machines copy their instructions into empty cells.

While these early studies describe structures that self-replicate, these structures generally consist of tens of thousands of components or active cells, and their self-replication has thus never actually been simulated computationally because of their tremendous size and complexity. Only recently has a simplified version of von Neumann's universal computer-constructor actually been implemented [51]. This implementation involved redesigning many of the components and extending the original transition function. Self-replication with this new universal computer-constructor remains to be demonstrated; this will require design of a tape that encodes a description of the universal computer-constructor (Pesavento, 1997, personal communication).

3. SELF-REPLICATING LOOPS

The complexity of even the simplified early cellular automata models described above seems consistent with the remarkable complexity of biological self-replicating systems: they appear to suggest that self-replication is, from an information processing perspective, an inherently complex phenomenon. Recent work with self-replicating loops provides evidence that this is not necessarily so, and represents a major step forward in efforts to produce simpler self-replicants.

In this section, we consider sheathed and unsheathed replicating loops, and discuss some issues concerning component symmetry and how simple self-replicating structures can be. For clarity and preciseness, in the remainder of this article, self-replicating structures are labeled by their type (SL = sheathed loop, UL = unsheathed loop, PS = polyominoe structure) followed by the number of components, the rotational symmetry of the individual cell states (S=strong, W=weak; explained below), the number of possible states a cell may have, and the type of neighborhood (V=von Neumann, M=Moore). For example, the sheathed loop discussed next is labeled SL86S8V because it spans 86 active cells, has strongly-symmetric cell states with each cell assuming one of 8 possible states, and its transition function is based on the 5-neighborhood (von Neumann neighborhood). This labeling convention provides a compact description of the loops we consider.

3.1 Sheathed Loops

A much simpler self-replicating structure based on 8-state cells, the sheathed loop, was developed by Langton in the mid-1980's (see Fig. 5b) [33]. The term "sheathed" here indicates that this structure is surrounded by a covering or sheath (X's in Fig. 5a-c). Before examining self-replicating loops, first consider Fig. 5a where a non-replicating loop plus arm (the latter coming off the lower right of the loop) is shown. The loop consists of a core of cells in state O and a sheath of cells in state X. In this case, a signal + followed by a blank space (quiescent cell) circulates around the "circular" data path forming the loop. Rules similar to those governing signal propagation in the data path of Fig. 4 act here to support the counter-clockwise circulation of signals. Each time the signal reaches the lower-right branch point where the arm extends from the loop, a copy of it passes out the arm. This non-replicating loop can be viewed as a storage element (any signal sequence circulating in it represents stored information), and similar non-replicating structures were used as parts in the universal computer-constructors designed by von Neumann and Codd.

a.
```
 XXXXXXXX
XOOOOOOOOX
XOXXXXXXOX
XOX    XOX
XOX    XOX
XOX    XOX
XOX    XOX
XOXXXXXXOXXXXX
XOO +OOOOOOOOOX
 XXXXXXXXXXXXX
```

b.
```
 XXXXXXXX
XO+ OL OLX
X XXXXXX X
X+X    XOX
XOX    XOX
X X    XOX
X+X    XOX
XOXXXXXXOXXXXX
X +O +O +OOOOOX
 XXXXXXXXXXXXX
```

c.
```
 XX
XLOX
XL+X
 X*
```

d.
```
-O+-O+-OL-OL
+          -
O          O
-          O
+          O
O          O
-          O
+          O
O          O
-          O
+          O
O-+O-+O-+O-+OOOO
```

e.
```
O+-OL-OL
-      -
+      O
O      O
-      O
+      O
O      O
-+O-+O-+OOOO
```

f.
```
OOO
O O
L++OO
```

g.
```
OO
L+OO
```

h.
```
OO<OO<LLOOOO
v          O
O          O
O          O
v          O
O          O
O          O
v          O
O          O
O          O
v          O
OO>OO>OO>OO^OOOO
```

i.
```
OO<LLOOO
v      O
O      O
O      O
v      O
O      O
O      O
>OO>OO>OOOOO
```

j.
```
OOO
O O
L>>OO
```

k.
```
OO
L>OO
```

Figure 5: Self-replicating loops in two dimensional cellular automata. Cells in the quiescent state are indicated by blank spaces. (a) Sheathed but non-replicating loop. A core of O's is surrounded by a sheath of X's. A single signal (+ followed by blank space) repeatedly circulates counterclockwise around the loop. (b) A self-replicating sheathed loop designated SL86S8V; (c) A small self-replicating sheathed loop SL12S6V; (d)-(g) Unsheathed self-replicating loops designated UL48S8V, UL32S8V, UL10S8V, and UL06S8V, respectively; (h)-(k) Unsheathed self-replicating loops designated UL48W8V, UL32W8V, UL10W8V, and UL06W8V, respectively.

Fig. 5b shows the initial state of a self-replicating *sheathed loop* that, as noted above, we will designate as SL86S8V [33]. The signal or instruction sequence + + + + + + L L that directs replication is embedded in the core of O's forming a loop similar to that shown in Fig. 5a (reading clockwise around the loop starting at the lower right corner). As copies of this circulating signal sequence periodically reach the end of the arm, they trigger the growth and turning of that arm to form a duplicate loop in the nearby cellular space, as explained in more detail below.

In creating a sheathed loop that replicates, the biologically-implausible requirements of universal computability and of ability to function as a universal constructor that were used in earlier models were abandoned. To avoid certain trivial cases, replicating loops are required to have a readily-identifiable stored instruction sequence or program that is used by the underlying transition function in two ways: as instructions that are interpreted to direct the construction of a replica, and as uninterpreted data that is copied onto the replica [33]. Thus, self-replicating loops are truely "information replicating systems" in the sense that this term is used by organic chemists [46].

The original sheathed loop was a modified version of a periodic emitter, a storage element and timing device in Codd's model [11]. Whereas von Neumann had used unsheathed data paths similar to that in Fig. 4, Codd introduced the analogous concept of a sheathed data path in his model. This consists of a series of adjacent cells in state O called the *core* covered on both sides by a layer of cells in state X called the *sheath* similar to what is illustrated in Fig. 5a. The sheathed data path served as a means for signal propagation, where signals or instruction sequences, represented by cells in other states embedded in the core of a data path, propagate along it. Codd's periodic emitter was a non-replicating loop similar to that in Fig. 5a except that it contained a more complex sequence of signals that continuously circulated around the loop. Each time the signal sequence passed the origin of the arm (lower right of loop in Fig. 5a) a copy of the signal would propagate out the arm and, among other things, could cause the arm to lengthen or turn. Langton showed that Codd's sheathed loop, a part of a much larger self-replicating structure, could be made self-replicating all by itself by storing in it a set of instructions that direct the replication process [33]. The "program" of the replicating sheathed loop, pictured in Fig. 5b, consists of individual instructions $+$, meaning "extend the current data path one cell", and *LL*, meaning "extend and turn left". Thus, the sheathed loop's instruction sequence ++++++LL can be interpreted as "extend the data path forward seven cells, then turn left". As this instruction sequence passes out the loop's arm it is "executed" as it reaches the end of the arm or growing structure. Each time the instructions are executed they generate one side of a new loop. Thus, executing these instructions four times causes the arm to repeatedly extend and turn until a second loop is formed, detaches, and also begins to replicate, so that eventually a growing "colony" of self-replicating loops appears. This replicating sheathed loop consists of 86 active cells as pictured in Fig. 5b, and its transition function has 207 rules based on the 5-neighborhood.

Subsequently, two smaller self-replicating sheathed loops containing as few as 12 active cells in one case were described (Fig. 5c) [6].

3.2 Unsheathed Loops

Following Codd's and Langton's work, we hypothesized that a number of alterations could be made that would result in even simpler and smaller self-replicating structures [57]. Such simplification is important for understanding the minimal information processing requirements of self-replication, for relating these formal models to theories of the origins of life, and for identifying configurations so simple that they might actually be synthesized or fabricated. One potential simplifying alteration is removal of the sheath surrounding data paths. It was not obvious in advance that complete removal of the sheath would be possible. The sheath was introduced by Codd and retained in developing sheathed loops because it was believed to be essential for indicating growth direction and for discriminating right from left in a strongly rotation-symmetric space ([11], p.40; [6], p. 296). In fact, we discovered that having a sheath is not essential for these tasks, and its removal leads to smaller self-replicating structures that also have simpler transition functions.

To understand how the sheath (surrounding covering of X's) can be discarded, consider the unsheathed version UL32S8V (shown in Fig. 5e) of the original 86-component sheathed loop (shown in Fig. 5b). The cell states and transition rules of this unsheathed loop obey the same symmetry requirements as those of the sheathed loop, and the signal sequence +–+–+–+–+–+–L–L– directing self-replication is the exact same program written using different "instruction codes" (+– for "extend", *L*– for "extend and turn left"). As illustrated in Fig. 6, just as with sheathed loops the instruction sequence circulates counterclockwise around the loop, with a copy passing onto the construction arm. As the elements of the instruction sequence reach the tip of the construction arm, they cause it to extend and turn left periodically until a new loop is formed. A "growth cap" of X's at the tip of the construction arm enables directional growth and right-left discrimination at the growth site (seen in Fig. 6b-d). It is this growth cap that makes elimination of the sheath possible. As shown in Fig. 6e, after 150 iterations or units of time the original structure (on the left, its construction arm having a.

```
                X
OL-OL-OO
-      O
+      O
O      O
-      +
+      -
O      O
-+O-+O-+O-+O
```

b.

```
OL-OOOOO
-      +
L      -
O      O
-      +
+      -
O      O    X
-+O-+O-+O-+OX
            X
```

c.

```
               X
-O+-O+-O      XO+-
+      L       X O
O      -         +
-      O         -
+      L         O
O      -         +
-      O         -
+O-+OOOOO-LO-LO-+O
```

d.

```
OL-OL-OO  -O+-O+-O
-      O  +      +
+      O XOX     -
O      O  X      O
-      +         L
+      -         -
O      O         O
-+O-+O-+O-+OOOOO-L
```

e.

```
       O
       +
       -
       O
OOOO+-O+  O+-OL-OL
O      -  -      -
-      O  +      O
L      +  O      O
O      -  -      O
-      O  +      O
L      +  O      O
O-+O-+O-  -+O-+O-+OOOO
```

Figure 6: Successive states of unsheathed loop UL32S8V starting at time t=0. The instruction sequence repeatedly circulates counterclockwise around the loop with a copy periodically passing onto the construction arm. At t=3 (**a**) the sequence of instructions has circulated 3 positions counterclockwise with a copy also entering the construction arm. At t=6 (**b**) the arrival of the first + state at the end of the construction arm produces a growth cap of X's. This growth cap, which is carried forward as the arm subsequently extends to produce the replica, is what makes a sheath unnecessary by enabling directional growth and right-left discrimination even though strong rotational symmetry is assumed (see text). Successive arrival at the growth tip of +'s extends the emerging structure and arrival of L's causes left turns, resulting in eventual formation of a new loop. Intermediate states are shown at t=80 (**c**) and t=115 (**d**). By t=150 (**e**) a duplicate of the initial loop has formed and separated (on the right); the original loop (on the left, construction arm having moved to the top) is beginning another cycle of self-directed replication.

moved to the top) has created a duplicate of itself (on the right).

This unsheathed loop UL32S8V not only self-replicates but it also exhibits all of the other behaviors of the sheathed loop: it and its descendents continue to replicate, and when they run out of room for new replicas, they retract their construction arm and erase their instruction code. After several generations a single unsheathed loop has formed an expanding "colony" where actively replicating structures are found only around the periphery. Unsheathed loop UL32S8V has the same number of cell states, neighborhood relationship, instruction sequence length, rotational symmetry requirements, and so on, as the original sheathed loop and it replicates in the same amount of time. However, it has only 177 rules compared to 207 for the sheathed loop, and is less than 40% of the size of the original sheathed loop (32 active cells versus 86 active cells, respectively). The rules forming the transition function for UL32S8V are given in [58].

Successful removal of the sheath makes it possible to create a whole family of self-replicating unsheathed loops using 8-state cells and strongly rotation-symmetric cell states. Examples of these self-replicating structures are shown ordered in terms of progressively decreasing size in Fig. 5d-g (labeled UL48S8V, UL32S8V, UL10S8V, and UL06S8V, respectively) and are summarized in the first four rows of Table 2. Each of these structures is implemented under exactly the same assumptions about the number of cell states available (eight), rotational symmetry of cell states, neighborhood, isotropic and homogeneous cellular space, and so forth, as sheathed loops within Codd's framework [11]. Given the initial states shown here, it is a straightforward but tedious and time-consuming task to create the transition rules needed for replication of each of these structures [58]. The smallest unsheathed loop in this specific group using 8-state cells, UL06S8V in Fig. 5g, is listed in line 4 of Table 2; it is more than an order of magnitude smaller than the original sheathed loop (SL86S8V; line 9 of Table 2). Consisting of only six components and using the two instruction sequence +L, it replicates in 14 units of time (column Replication Time in Table 2). Replication time is defined as the number of iterations it takes for both the replica to appear and for the original structure to revert to its initial state. This very small structure uses a total of 174 rules (Total Rules in Table 2) of which only 83 are needed to produce replication (Replication Rules); the remaining rules are used to detect and handle collisions between different growing loops in a colony, and to erase the construction arm and instruction sequence of loops during the formation of a colony. If one counts only those rules which cause a change in state of the cell to which they are applied, this structure uses a total of 91 rules (State Change Rules) of which only 49 are used to produce replication (State Change Replication Rules). This latter measure is taken here to be the preferred measure of the information processing complexity of a transition function because it includes only rules needed for replication and only rules that cause a state change.

Table 2: Replication Time and Number of Rules

Label	Replication Time	Total Rules	Replication Rules	State Change Rules	State Change Replication Rules	Reduced Total Rules	Reduced Replication Rules
UL48S8V	234	177	167	109	104	75	72
UL32S8V	150	177	166	109	104	74	71
UL10S8V	34	163	117	74	54	50	40
UL06S8V	14	174	83	91	49	66	32
UL48W8V	234	142	98	80	52	68	42
UL32W8V	151	134	98	77	52	66	42
UL10W8V	34	114	82	43	35	31	24
UL06W8V	10	101	58	44	31	33	20
SL86S8V	151	207	181	118	101	90	77
UL32S6M	150	-	305	-	129	-	-
UL10W8M	34	-	221	-	56	-	-
UX10W8V	44	173	103	70	36	57	25
SL12S6V	26	145	140	61	60	46	45
UL06S6V	18	115	83	64	46	30	30
UL05S6V	17	65	58	35	35	23	23

Prior to [57], the smallest previously described structure that persistently self-replicates under the same assumptions [6], designated SL12S6V here, uses 6-state cells, has 12 components (Fig. 5c), and as indicated in Table 2, requires 60 state change replication rules. We have been able to create unsheathed loops, designated UL06S6V and UL05S6V, using 6-state cells with half as many components and requiring only 46 or 35 state change replication rules, respectively (last two rows of Table 2). The initial state of UL06S6V is shown in Fig. 5g, and that of UL05S6V is identical except it has one less component in its arm; the complete transition functions are given in [58]. To our knowledge, UL05S6V is the smallest and simplest self-replicating structure created under exactly the same assumptions about cell neighborhood, symmetry, and so forth, as sheathed loops.

3.3 Varying Rotational Symmetry

Cellular automata models of self-replicating structures have usually assumed that the underlying two dimensional space is homogeneous (every cell is identical except for its state) and isotropic (the four directions NESW are indistinguishable). However, there has been disagreement about the desirable rotational symmetry requirements for individual cell states as represented in the transition function. The earliest cellular automata models, such as von Neumann's, had transition functions satisfying weak rotational symmetry: some cell states were directionally oriented [5, 66, 70]. These oriented cell states were such that they permuted among one another consistently under successive 90° rotations of the underlying two-dimensional coordinate system.[1] For example, the cell state designated $\uparrow$ in von-Neumann's early work is oriented and thus permutes to different cell states $\rightarrow$, $\downarrow$, and $\leftarrow$ under successive 90° rotations; it represents one oriented component that can exist in four different states or orientations. However, Codd's simplified version of von Neumann's self-replicating universal constructor-computer [11] and the simpler replicating sheathed loops [33] are based upon more stringent criteria called strong rotational symmetry. With strong rotational symmetry all cell states are viewed as being unoriented or rotationally symmetric. The transition functions for the unsheathed loops shown in Fig. 5d-g also all use this strong rotational symmetry requirement (indicated by S in their labels). Their eight cell states are designated

$$.O\#L - *X+$$

[1]A formal definition of rotational symmetry in cellular automata can be found in [11]. Care should be taken not to confuse the rotational symmetry of a cell state as interpreted by the transition function with the rotational symmetry of the printed character used to represent that state. Here the printed character L is not rotationally symmetric, for example, but the cell state it represent is treated as such.

where the period indicates the quiescent state. All of these states are treated as being unoriented or rotationally symmetric by the transition function.

The fact that the simplest self-replicating structures developed in the past [11, 33] were all based on strong rotational symmetry raises the question of whether the use of unoriented cell states intrinsically leads to simpler algorithms for self-replication. Such a result would be surprising as the components of self-replicating molecules generally have distinct orientations. To examine this issue we developed a second family of self-replicating unsheathed loops, with examples shown in Fig. 5h-k (labeled UL48W8V, UL32W8V, UL10W8V, and UL06W8V, respectively), whose initial state and instruction sequence are similar to those already described in Fig. 5d-g. However, for the structures in Fig. 5h-k weak symmetry is assumed, and the last four of the eight possible cell states

$$.O\#L\wedge > \vee <$$

are treated as oriented. In other words, although there are still 8 states, the cell state $\wedge$ is considered to represent a single component that has an orientation and thus can exist pointing up or in the three other directions $>$, $\vee$ and $<$. The remaining four cell states (. 0 # L) are unoriented. For example, in Fig. 5i the states $>$, $\vee$, and $<$ appear on the lower, left and upper loop segments, respectively, to represent the instruction sequence $<<<<<< LL$. While cells in such a model have 8 possible states and are thus comparable in this sense with the above work on sheathed and unsheathed loops (Fig. 5a-g), they also can be viewed as simpler in that they have only 5 distinct possible components. As can be seen in Table 2 (lines 5-8), where the presence of oriented cell states or weak symmetry is indicated by W in the structure labels, relaxing the strong rotational symmetry requirement like this consistently leads to transition functions requiring fewer rules than the corresponding strong symmetry version; this is true by any of the measures in Table 2. This decrease in complexity occurred in part because the directionality of the oriented cell states intrinsically permits directional growth and right-left discrimination, making even a growth cap unnecessary.

This simplicity and speed of replication made possible by weak rotational symmetry are

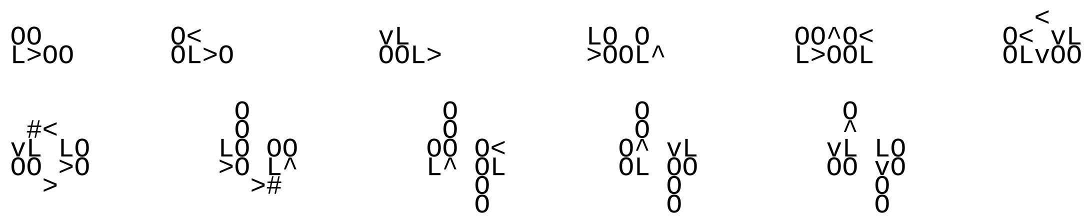


Figure 7: Structure UL06W8V uses only five unique components. Shown here are eleven immediately successive structures ordered left to right, top to bottom. Starting at $t = 0$, the initial state shown at the upper left passes through a sequence of steps until at $t = 10$ (last structure shown) an identical but rotated replica has been created.

illustrated in Fig. 7 where the complete first replication cycle of UL06W8V is shown. Only 31 rules are needed to direct replication of this small structure which makes use of only 5 possible components. After several generations the older, inactive structures are surrounded by persistently active, replicating progeny, as shown in Fig. 8, and this colony formation continues indefinitely. The small but complete set of transition function rules needed for one replication of UL06W8V can be found in [57].

The results summarized in Table 2 lead to additional observations about unsheathed loops [57]. For systems with either weak or strong symmetry requirements, the number of rules in the transition function required for replication increases as structure size increases but then levels off to a value characteristic of which of the symmetry requirements are in effect. Replication time is essentially independent of the type of rotational symmetry used (strong versus weak) but is proportional to the size of the self-replicating loop. This proportionality is effectively linear. To assess the effects of type of neighborhood, we implemented versions of the two arbitrarily-selected unsheathed loops shown in Fig. 5e and 5j using the 9-neighborhood (Moore neighborhood). The resultant systems, designated UL32S6M and UL10W8M in Table 2, had the same replication time as identically-structured loops UL32S8V and UL10W8V but required dramatically more rules in their transition functions for replication.

3.4 Reduced Rule Sets

As noted earlier, the complete transition function for self-replicating loops includes a number of rules that are extraneous to the actual self-replication process (such as instruction

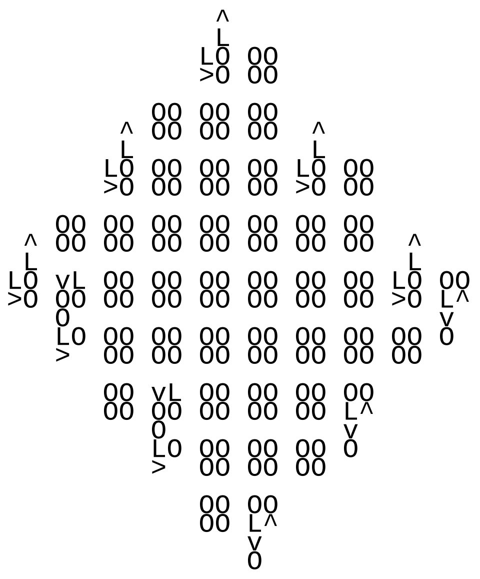


Figure 8: After several generations, a colony has formed from the original single copy of structure UL06W8V pictured in the preceding figure. Structures around the periphery are still actively replicating; those in the center have retracted their arms and erased the instruction sequence that directs their self-replication. Growth of this colony continues indefinitely (this was verified by computer simulations out to at least 11 generations for all of the unsheathed loops described in this article).

sequence erasure) and many rules which simply specify that a cell state should not change. The *state change rules*, the subset of rules that specify that a cell's state should change, alone are completely adequate to encode the replication process. As noted above, we believe that the number of state change rules used for one replication is thus the most meaningful measure of complexity of transition functions supporting self-replication. As shown in the sixth column of Table 2, this measure indicates that, from an information processing perspective, algorithms for self-directed replication can be relatively simple compared to what has been recognized in the past, especially when oriented components are present.

The simplicity of unsheathed loop transition functions when oriented components are used is even more striking if one permits the use of unrestricted placeholder positions in encoding their rules. We implemented a search program that takes as input a set of rules representing a transition function, and produces as output a smaller set of reduced rules containing "don't care" or "wildcard" positions [58]. The size of the reduced rule sets that result from applying this program to the complete original set of rules and to only the replication rules of each of the cellular automata models described above is shown in the rightmost two columns of Table 2. With UL06W8V this procedure reduces the complete rule set from 101 to 33 rules, and the set of rules needed for one replication from 58 to 20. Thus, by capturing regularities in rules through wildcard or "don't care" positions, it is possible to encode the replication process for unsheathed loop UL06W8V in only 20 rules. Computer simulations verified that these 20 rules can guide the replication of UL06W8V in exactly the same way as do the original rules. As shown in Table 2, similar reductions occur with other self-replicating structures. Such simple systems indicate that self-replication can in principle be far simpler than previously recognized.

4. EMERGENCE OF SELF-REPLICATION

The self-replicating structures described so far have all been initialized with an original copy of the structure that will replicate (the "seed") and have been based on manually created transition rules designed for that single, specific structure. Recently, we have taken a new direction in creating self-replicating structures, focusing on self-replication as an emergent property. In this section we give two examples of our work in this area. The first example shows an approach where no initial replicants are present. Instead, self-replicating structures emerge from initial states consisting of random isolated components. The second example shows how, given a small but arbitrary initial structure, a genetic algorithm can be used to automatically discover a set of transition rules that will cause that structure to replicate.

4.1. Emergence of Replicators

Recent work by our group has shown that it is possible to create cellular automata models in which a simple self-replicating loop emerges from an initial state having a random density and distribution of components (the "primordial soup") [8]. These emergent self-replicating loops employ a general purpose rule set that supports the replication of loops of different sizes and their growth from smaller to larger ones. This rule set also allows random changes of loop sizes and interactions of self-replicating loops within a cellular automata space containing free-floating components.

An example running in a randomly initialized, small (40×40) cellular automata space using an initial component density of 25% is shown in Figure 9. Periodic boundary conditions are used (opposite edges are taken as connected), so the space is effectively a torus. Initially, at time $t = 0$ (upper left of Figure 9), the space is 25% filled by randomly placed, non-replicating components designated as O, $>$, or L, while cells in the quiescent state are indicated by blank spaces. All components have strong rotational symmetry except $>$ which is viewed as being oriented, as described above.

This simulation is characterized by the initial emergence of very small, self-replicating loops and their progressive evolution to increasingly large and varied replicants. During this process a replicating loop may collide with other loops or with free-floating components, and either recover or self-destruct. Thus, by time 500 (upper right of Figure 9), very small self-replicating loops of size 2×2 and 3×3 are present. By time 1500 a 4 x 4 loop is about to generate a 5 x 5 loop in the middle left region. At time 3000 the biggest loop is 8 x 8 and it is about to generate a 9 x 9 loop. By time 5000 many very large loops have annihilated each other and only one intact 10 x 10 loop is left. By time 7500 all large loops have "died", but there are new 3 x 3 loops in the space. These loops will replicate and it is not clear when (if ever) self-replication will cease. In this example, the size of the replicating structures became too big to fit comfortably in such a small world (40×40 only), and the large loops tended to annihilate each other.

As can be seen from this example, the transition function supporting these self-replicating loops differs from those used in previous cellular automata models of self-replication in several ways. A self-replicating structure emerges from an initial random configuration of components rather than being given, replication occurs in a milieu of free-floating components, and replicants grow and change their size over time, undergoing annihilation when replication is no longer possible. All of this occurs in the presence of a single transition function based on the 9-neighborhood (Fig. 1). As is increasingly being done in cellular automata modeling, the

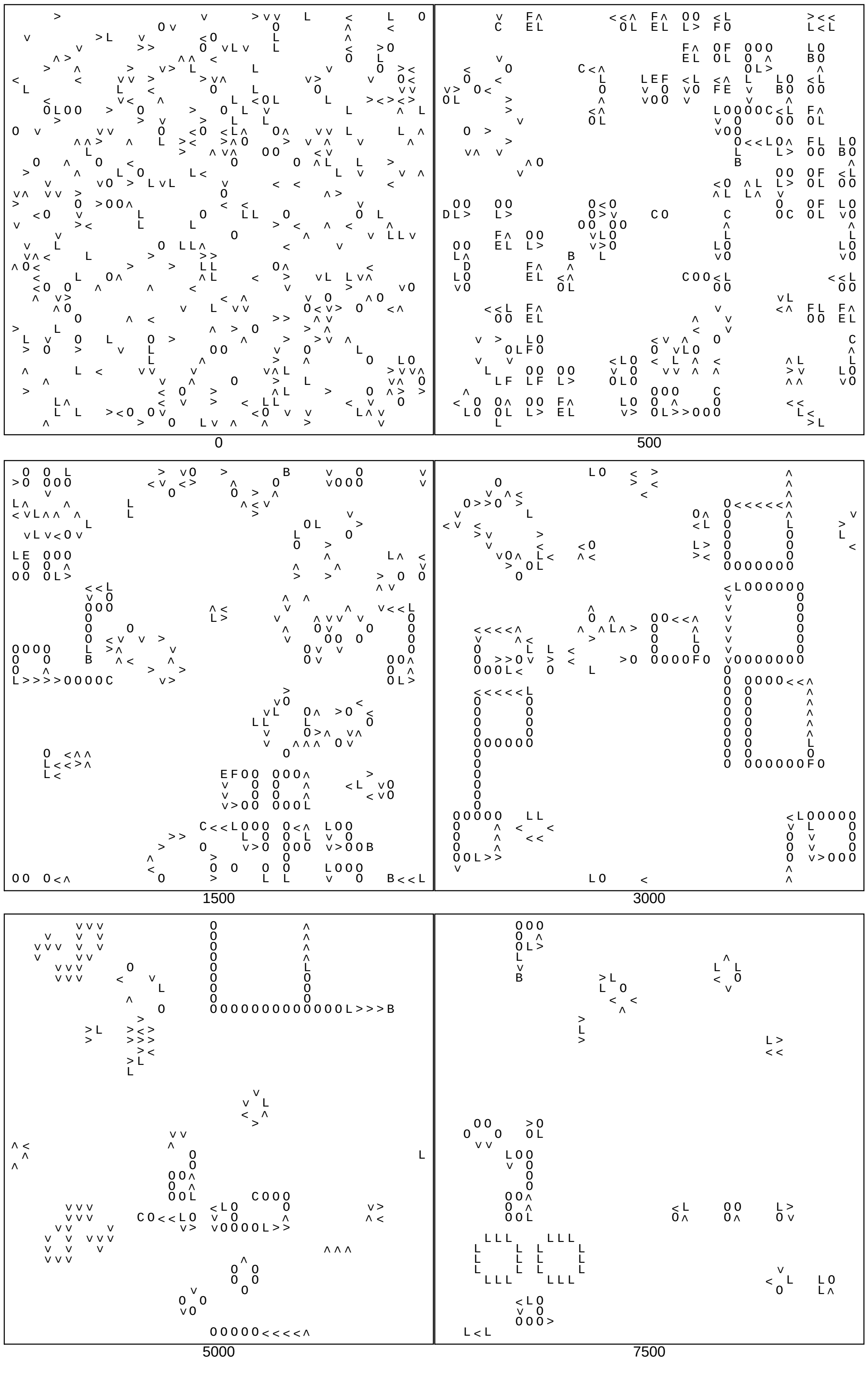
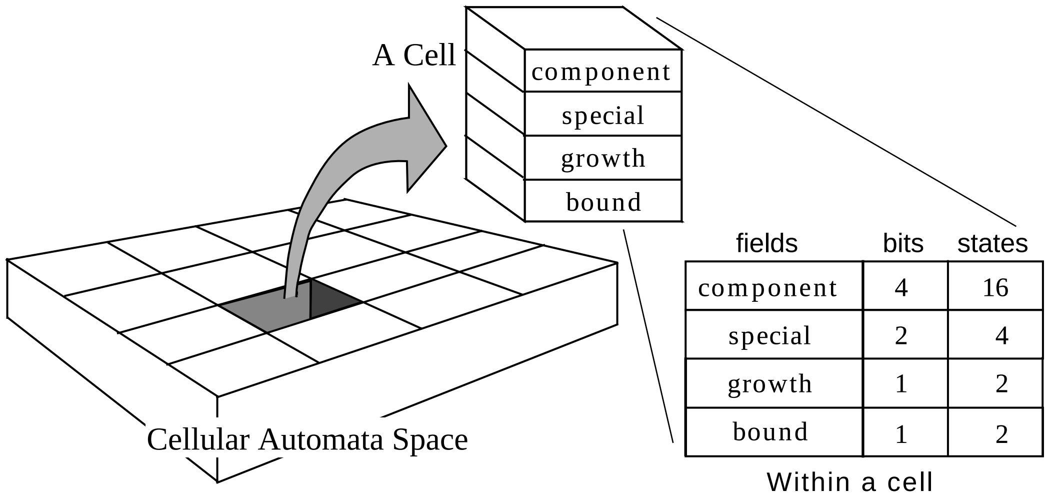


Figure 9: A running example of emergent self-replication. Times are shown.

transition function is based on a functional division of data fields [67]. As seen in Figure 10, the bit depth of a cellular automata cell (in our case 8 bits) is functionally divided into four different *fields* (4, 2, 1 and 1 bits each) such that each field encodes different meanings and functions to the rule writer. The utilization of field divisions greatly simplifies the cellular automata rule programming effort, and makes the resulting rules much more readable. In the illustrations in this paper, only the component field is shown.



fields	bits	states
component	4	16
special	2	4
growth	1	2
bound	1	2

Figure 10: The 8 bit state variable in each cell is conceptually sliced into four different bit groups called *fields*. Each field represents a specific piece of information.

As noted earlier, each non-quiescent or active cell is taken to represent a potential "component" of a cellular automata structure. A cellular automata structure can be just a single cell, i.e., one with no conceptual connection with any adjacent non-quiescent cells, and in that case we call it an *unbound component.* On the other hand, a cellular automata structure can consist of several contiguous non-quiescent cells that are functionally interrelated, behaving as a whole, such as a self-replicating loop. In the latter case we call the structure a multi-component structure or simply a structure, and we call its components *bound components* (their *bound bit* is set; see Figure 10).

The four data fields (Figure 10) and their states in the transition function are as follows. The four-bit *component field* accounts for most normal operations of cellular automata structures. It encodes twelve state values (out of 16 possible) corresponding to components just as in the previous examples we have seen. These include O (building block of data paths), > (signals growth of data path; this actually represents four states), B (birth of new component), L (left turn signal), C (corner), and D, E, F (branching/detachment). There is also the quiescent state which is as usual shown as white space in all figures. The other fields are new. A two-bit *special field* denotes special situations that arise occasionally in the cellular automata space, such as branching, blocking passage of signals on a data path, or dissolution of a loop. A one-bit *growth field*, if set, marks a stimulus that may cause the existing signal sequence to increase in length. A one-bit *bound field*, if set, marks a cell as part of a multi-cell structure; otherwise the cell is an unbound component.

The complete set of rules forming the transition function support replication of loops in a fashion similar to those used in the past [33, 57]. In addition, a loop's replicant can be of a different (larger) size, a process referred to as *extended replication.* A loop's signal sequence can become modified to generate loops larger than itself if by chance an active growth field appears in one of its cells during the arm branching process. Cellular automata rules that support extended replication are new. In the past, a different rule set has been required for each size replicating loop; here the emergence of different size loops and their simultaneous replication is supported by a single rule set. This permits an initially small emergent self-replicating structure to grow in size.

Another new aspect of this model is collision detection and resolution. In all past work on self-replicating loops, replication occurs in an otherwise empty space and the transition function does not need to handle unanticipated events. In other words, while writing the rules one has complete control over the behaviors occurring in the cellular automata space, including the initial state. In contrast, here the very first assumption is that there is no *a priori* knowledge about the interactions between self-replicating loops, or what the cellular automata space is like at time zero. Although the rules in the previous models of replication that we have considered so far can reliably direct a structure to do replication in isolation, they cannot guarantee that a structure will not run into another structure, that two structures will not try to replicate into the same region of the cellular automata space, or that a replicating loop will not run into free-floating unbound components. These factors are all "randomly" determined. The transition function used here thus assumes that not all designated regular procedures will always be followed without interruption or disturbance from other structures. It includes rules that will detect failed procedures and clean up the cellular automata space after such failures. When a loop has any of its cells enter a failure mode, this mode quickly spreads throughout the whole structure, causing the loop to *dissolve* completely. The loop's components become unbound and revert to being controlled by the rules governing unbound components.

There is no *a priori* information about when and where growth bits should be placed in this model of emergent replication, and none are set initially. In the example shown here, whenever a signal L dissolves or "dies", it leaves behind a growth bit at its location. A loop usually has only one L signal, so one dissolving loop usually produces one new growth bit in the cellular automata space. This way, the generation of the growth bit becomes part of the behavior of the cellular automata space, since when and where a loop will dissolve is determined purely by the interactions within the cellular automata space. The growth bit is utilized during the arm branching phase of a self-replicating loop to extend the signal sequence in a loop. As shown in Figure 11, this is a two step strategy. First, if a signal $>$ sees a growth bit in its place and it is the last $>$ before the signal L, it does *not* copy the signal L behind itself as it normally does. Instead, it stays at its current value $>$ for one more time step, thus effectively increasing the size of the signal sequence by one. The signal L disappears temporarily since it is not copied, but reappears when the signal $>$ sees a trailing signal F and the growth bit in its position. The growth bit is unset after the signal L is regained, so the same growth bit does not cause another growth stimulus. Thus, when a loop dies, it leaves a set growth bit behind, and when a loop expands, it consumes a growth bit. This provides an interesting ecological balancing factor in the cellular automata universe.

The emergence of self-replication is achieved by allowing the unbound components to translate and change or appear at "random", i.e., by "stirring the primordial soup", until the configuration corresponding to a small (2 x 2) loop occurs by chance. The rules that do this can be summarized by:

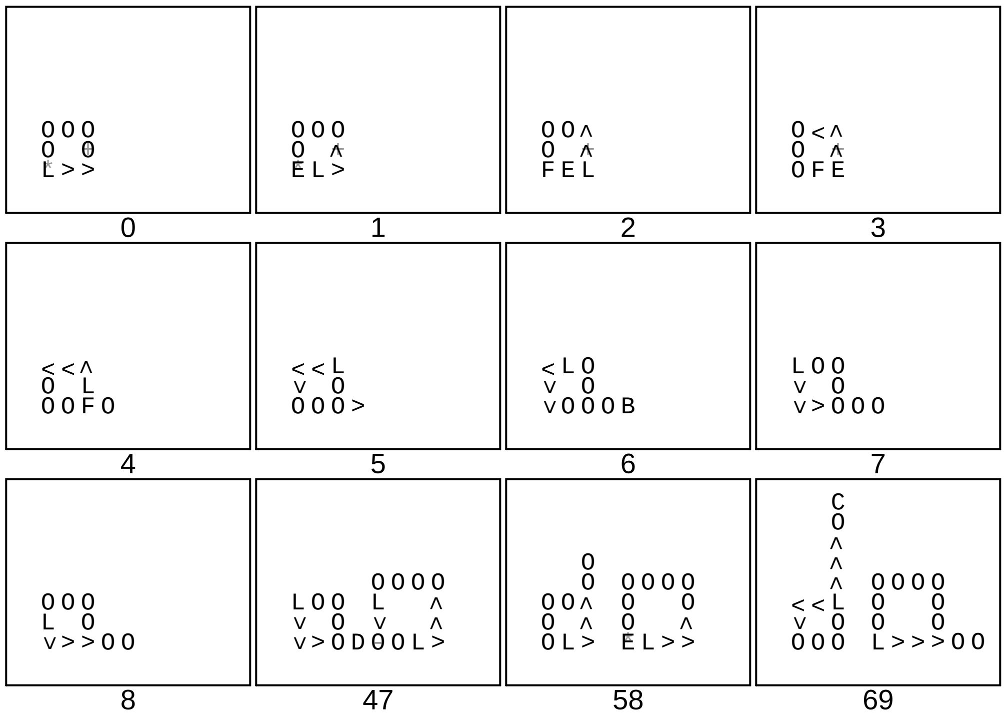


Figure 11: The growth of a larger loop (extended replication). At time 0 the branch special flag in the lower left cell and the growth bit in the middle right cell are both set. At time 2 the normal arm branching EF signal sequence is generated. At time 3 the signal sequence becomes >>> and subsequently the growth bit is unset. By time 8 the parent loop is about to start the replication cycle with one more > signal than it normally has. By time 47 a whole new loop bigger than the original one is generated. By time 58 the two loops have separated and the original one is just about to start another replication cycle. At time 69 the new, larger loop is finished and is starting its own replication cycle.

- If a quiescent cell has exactly three active neighbors, it becomes active at the next time step. Its active value is determined based on the state of its neighbors.
- If an active cell has exactly two or three active neighbors, it will stay active; otherwise, an active cell will return to the quiescent state at the next time step.

These rules, are generalizations (from binary to non-binary states) of those used in the Game Of Life described earlier. These rules generally produce a continually varying distribution of unbound components.

All that is then required for the emergence of self-replication is a small set of rules that watch for the formation of the smallest loop configuration (a 2 x 2 loop). Once such a configuration occurs, all four members of it simultaneously set their own bound bit and produce an active smallest loop at the next time step. This is how the first self-replicant is formed. This is possible using only local operations because the minimum loop configuration is so small that it fits within a single 9-neighborhood, allowing each component to simultaneously "see" the same configuration. An example of how the unbound component rule set works and how it leads to the first self-replicating structure is demonstrated in Figure 12.

The behavior of this model of emerging self-replication has been examined experimentally [8]. Eighty one simulations were conducted while varying the cellular automata space size (50 x 50, 100 x 100, 150 x 150 and 200 x 200), initial unbound component density (10%, 20%, 30%, 40% and 50%) and random initial configuration used in each simulation. In 80 of these 81 simulations, self-replicating loops emerged, and usually these persisted indefinitely[2]. The emergence, proliferation and persistence of self-replicating loops were found to be robust phenomena relatively insensitive to the initial conditions of a simulation. There is a very stable and characteristic dynamics under the emergent self-replicating rule set. In fact, the number of active cells, and the fraction of bound/unbound components, always tended to approximate a long-term stable value. This value depends on an interaction between the rules governing

[2]The one simulation where self-replication did not occur was with the small, 50 x 50 space, where significant variations in unbound components ceased before the configuration of the smallest self-replicating loop appeared.

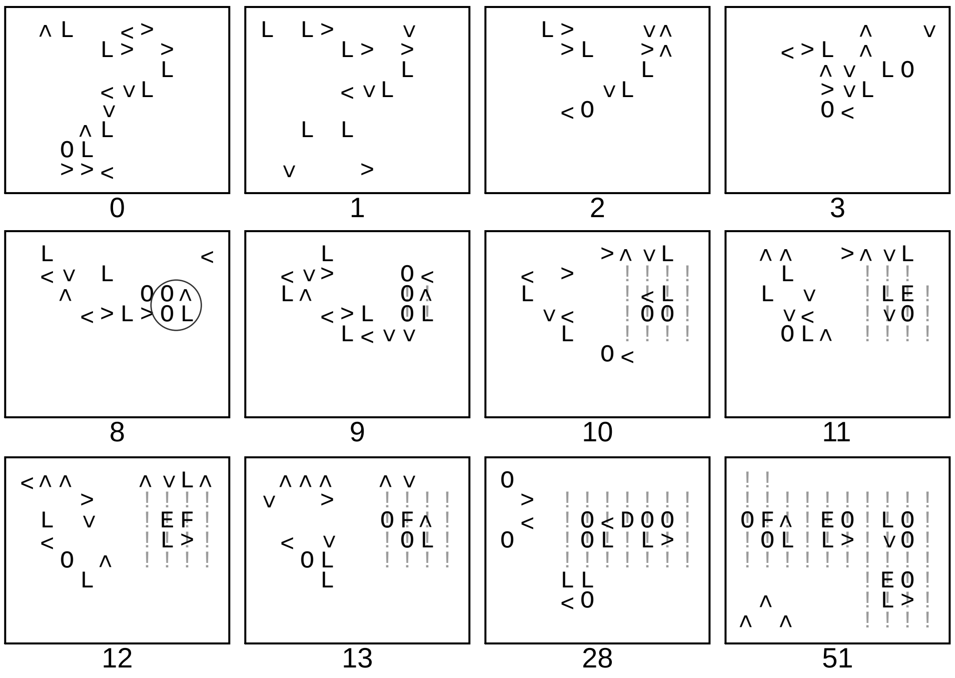


Figure 12: The emergence of a self-replicating structure. Components of structures are marked by a non-zero bound bit, or an '!' mark. At time 0 a randomly generated initial space is given. This space has only unbound components until time 8, when the pattern of the smallest replicating loop (circled) appears. At time 9 this configuration turns into a functioning self-replicating loop when its four cells set their bound bit simultaneously (set bound bits are indicated by faint exclamation points). Its peripheral cells clear and the arm branching process begins (times 10 to 13). By time 28 the first sibling is about to separate. By time 51 four loops are obtained and all are actively engaging in the replication processes.

replication and these governing movement of unbound components, and not on either of these subsets of rules alone. The number and size of replicating loops generally stabilizes too. After a few thousand time steps, there is typically no significant change in the average number and size of loops in the cellular automata space. These values tend to oscillate in a non-periodic, varying-amplitude fashion about a mean, suggesting an underlying chaotic dynamics. Details of these experiments can be found in [8]

These results show for the first time that non-trivial self-replicating structures can emerge in a cellular automata space initialized with a randomly distributed set of components. Some other computational studies of emergent self-replication have been done (see Chap. 28 of [31], and [48]), but these have not used cellular automata methods. For example, the investigation in [48] used a very different (non-cellular automata) model having an initial state composed of randomly generated sequences of computer operations. It evolved self-replication via a mutation operation. The primary conclusion, backed up by simulation results, was that the probability of a randomly generated sequence of operations becoming self-replicating increased with the number of computer operations it contained. Further, self-replicating sequences decreased in size once they appeared. The cellular automata model described here shows that such behaviors are not necessarily an inherent aspect of emergent self-replication, in that very small self-replicants can arise first and then increase in size, as is often argued to have occurred with the origins of biological replication. We attribute the differences in results to the fact that our cellular automata model starts with random individual components rather than random initial *sequences* of computer operations, that its rules were hand crafted, and that cellular automata are based solely on highly local operations (e.g., there is no global copy operation that copies a loop to a nearby region of the space).

4.2 Automatic Programming of Replicator Rules

In the past, the rules or transition function governing self-replicating structures have always been programmed manually. This is a very difficult and time-consuming task, and it is also influenced by subjective biases of the implementer. As an alternative, we have recently shown that it is possible to automatically generate a set of rules for self-replication, i.e., to automatically program a cellular automata space to generate a sequence of steps that the components of a structure can follow to produce replicants of the original structure [36, 37]. While work in this area is just beginning and the structures used so far are quite small, initial results have already created a new class of non-trivially replicating structures unlike those developed previously.

```
initialize population of chromosomes
evaluate fitness of each chromosome
while (termination criterion not reached) do
   select parent chromosomes for mating
   apply crossover and mutation to produce children
   evaluate fitness of each chromosome
end
```

Figure 13: Brief summary of traditional genetic algorithm.

The approach we describe here is based on using *genetic algorithms*, a powerful stochastic search technique, to discover rules producing self-replication. A genetic algorithm produces a solution to a problem by manipulation of a population of candidate solutions [19, 20, 24, 31, 42]. Each individual in the population, called a chromosome, encodes a potential solution to the problem under consideration. Typically the population is initialized with randomly generated chromosomes (see Figure 13). Each existing chromosome (problem solution) has its effectiveness as a solution to the problem measured by a fitness function. Then, simulating natural selection as it is understood to occur in biological evolution, the most highly fit chromosomes are selected to serve as parents for offspring chromosomes that form the next generation. This process occurs repeatedly with progressively better solutions being represented in the population. The genetic algorithm typically terminates after identifying a sufficiently good problem solution or after a prespecified number of generations.

A key aspect of this genetic search process is the use of genetic operators, such as crossover and mutation, in producing offspring chromosomes. Crossover takes two parent chromosomes and swaps randomly-selected parts of their contents to form two offspring chromosomes (Figure 14a). Mutation takes one parent/offspring chromosome and complements randomly-

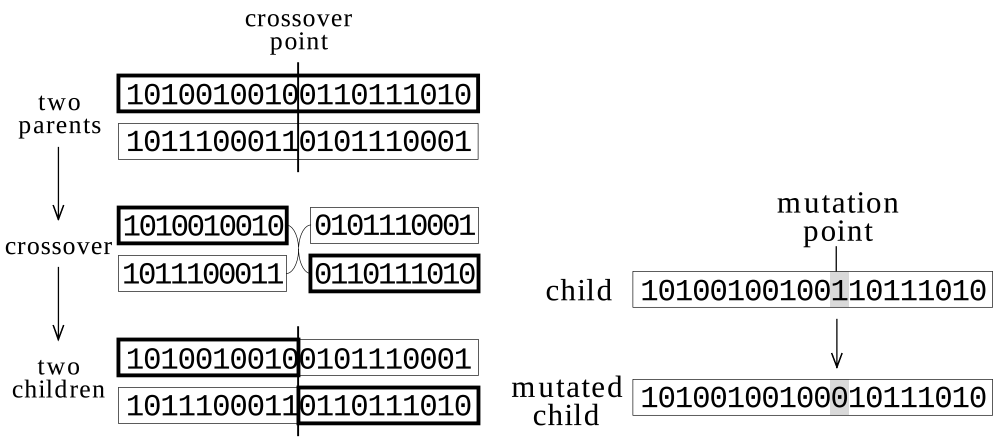


Figure 14: The two most commonly used genetic operators are crossover, illustrated on the left, and mutation, illustrated on the right. Each chromosome here is a binary string. These operations introduce variability into the chromosome manipulated by the genetic algorithm.

selected bits (Figure 14b). These alterations to the population of chromosomes, coupled with fitness-guided selection of parents, allows the genetic algorithm to heuristically and stochastically search the space of problem solutions for good solutions.

Relatively few previous studies have reported using a genetic algorithm to automatically produce rule tables for cellular automata (see, for example, [42, 59]. For self-replication, there are some clear barriers to using genetic algorithms in this way. One barrier is the enormous computational load involved. At each iteration of the genetic algorithm, a whole population of cellular automata models must be run, each individually involving a substantial amount of computation. This process must be done repeatedly, generation after generation, and the fitness of each individual rule table evaluated each generation. Further, the space of possible rules that must be searched is enormous. A second barrier is that it is not obvious what form a good fitness function should have. The straightforward approach of making fitness proportional to the number of replicants produced is generally useless. This is because there are typically no replicants produced by any randomly generated initial rule set in a population, so counting the number of replicants produced gives no guidance early on, reducing the genetic algorithm to blind search.

Fortunately, it has proven possible to solve these problems, at least to a limited extent

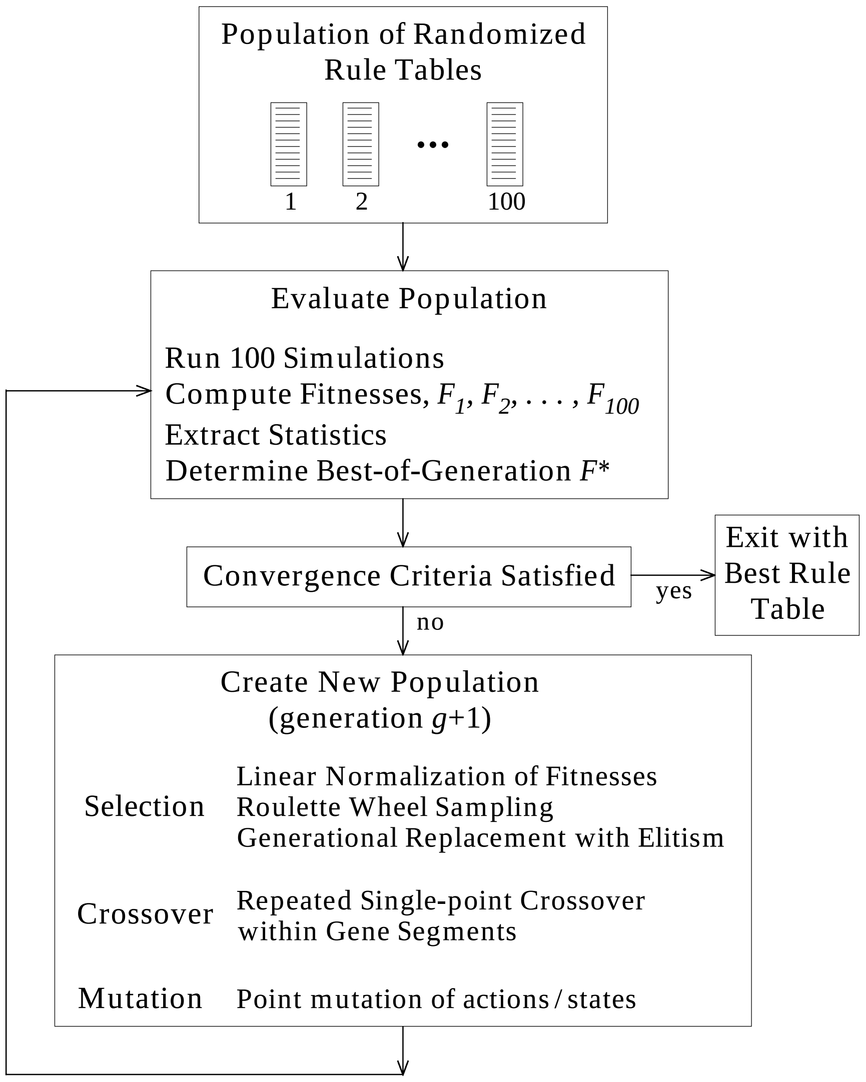

Population of Randomized Rule Tables
1
2
100
Evaluate Population
Run 100 Simulations
Compute Fitnesses, $F_1, F_2, \ldots, F_{100}$
Extract Statistics
Determine Best-of-Generation F^*
Convergence Criteria Satisfied
yes
Exit with Best Rule Table
no
Create New Population (generation g+1)
Selection
Linear Normalization of Fitnesses
Roulette Wheel Sampling
Generational Replacement with Elitism
Crossover
Repeated Single-point Crossover within Gene Segments
Mutation
Point mutation of actions / states

Figure 15: Schematic overview of the use of a genetic algorithm to search for a rule set that produces self-replication when given a simple but arbitrary initial structure in a cellular automata space. Given an arbitrary structure, an initial population of possible transition functions is randomly generated (top). In general, none of these initial rule sets will cause the given structure to replicate. Each rule table or transition function is then tested and its potential ("fitness") to be changed into rules that do produce self-replication is measured. If a rule set that does produce self-replication has been found, the program quits and returns that rule table. If not, a new population of rule tables is created by selecting the most promising or most fit existing rule tables. These promising rule tables are modified via crossover and mutation, and the entire process repeats.

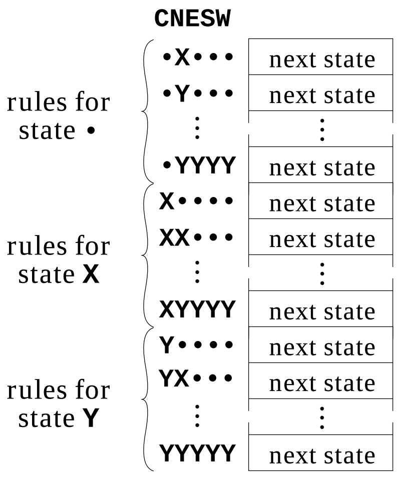


Figure 16: Encoding of a rule table used to represent a chromosome in the genetic algorithm of Fig. 15.

[36, 37]. Figure 15 summarizes a genetic algorithm that has been applied successfully to this task. The genetic algorithm begins by generating a population of randomly initialized rule tables, and uses these to execute cellular automata simulations, each starting with the same initial structure. Following these simulations, each rule table in the population receives a fitness measure F reflecting the degree to which its rules appear promising as a means of supporting self-replication. A new population is then created, randomly choosing rule tables to carry forward to the new population in proportion to their fitness. As the new population is formed, rule tables from the old population are "mixed together" and combined through the genetic operation of crossover, and randomly altered by mutation, as explained above. (An exception is that a copy of the very best rule table in a population is always carried forward unchanged.) At this point, the whole process iterates, this time starting with the new population of rule tables and discarding the old. Typical parameter values in a simulation like this include a population of 100 rule sets examined over 2000 generations, with probabilities of crossover and mutation of 0.8 and 0.1, respectively. At the end of this process, the most highly fit rule table is returned as a potential transition function supporting self-replication with the given initial structure.

Figure 16 shows the encoding of a rule table used by the genetic algorithm in this process, i.e., a chromosome representing one individual in the population. The rule table is indexed on the left by the 5-neighborhood pattern CNESW (center, north, east, south, west), and rules for each specific component are grouped together. Each rule has a "next state" entry indicating what the center cell component C should become at the next time step for the given neighborhood pattern. By adopting the convention that a rule for every possible neighborhood pattern must be represented in a chromosome, and that these are always in the same order, it is not necessary to explicitly store the CNESW neighborhood patterns. Thus a chromosome is represented as just a list of next-state entries (i.e., just the next state list indicated on the right in Fig. 16). For the simulations described below, chromosomes were roughly 850 next-state elements long.

Creating a fitness function F that accurately measures the promise of a rule table for eventually generating self-replication of an arbitrary initial structure was the most challenging aspect of this work. None of the initial random rule tables produce replicants, so in this sense each has a zero fitness. This issue was addressed by creating a fitness function F that is a linearly weighted sum of three measures,

$$F = w_g\ f_g\ +\ w_p f_p\ +\ w_r\ f_r,$$

where the w's are fixed weights ($0 < w < 1$) and the f's are fitness measures ($0 \leq f \leq 1$). The basic idea here is that an intermediate state on the path to evolving rules for self-replication is the evolution of a rule set that produces growth and/or configurations similar to that of the seed structure. Thus, the overall fitness F includes a *growth measure* f_g assessing the extent to which each component type in a given initial structure generates an increasing supply of that component from one time step to the next, and a relative *position measure* f_p assessing the extent that each component has the same neighbor components over time as it did in the initial structure. High values of f_g and f_p do not necessarily imply that replication is present (although replication, if present, would be expected to make them relatively large), but they do represent behaviors that might be useful precursors to replication. The third term in F, the *replicant measure* f_r, is a function of the number of actual replicants present. While this is zero for many early generations with a rule table, it can cause a substantial rise in F if actual replication occurs.

How should the three weights in F be chosen to maximize the chances of success with this approach? There is no precise answer that can be given to this question at present. Systematic experiments have suggested that $w_g = 0.05$, $w_p = 0.75$, and $w_r = 0.20$ is a good set of values when the weights are constrained to sum to 1.0 [37]. In other words, the relative positioning measure proved to be the most critical factor in discovering rules for self-replication.

To assess the success of the above approach, 100 experiments were done with each of several small arbitrary initial seed configurations. The rate of success in discovering rules producing self-replication declined sharply as the number of components in the initial structure increased. Under the best conditions, the percentage of runs in which the genetic algorithm discovered a rule-table that resulted in self-replication was 93% for structures with two components, 22% for structures with three components, and 2% for structures with four components.

A representative example of a self-replicating structure discovered in this fashion is shown in Figure 17. The naming convention used here to catalog these structures is similar to those for loops except the prefix PS is used (for "polyominoe structure") to designate the arbitrary block-like shape of the initial structure. Structure PS4W17V in Fig. 17 provides a typical example. It is a four-component replicator for which multiple replicants can be observed by t=5. Like self-replicating loops, these structures gradually form expanding colonies.

The replicators discovered in this fashion by the genetic algorithm can be viewed as forming a third class of self-replicating structures (the first two classes being complex universal computer-constructors and self-replicating loops). In addition to being formed from arbitrary non-loop seed structures, the replicators discovered in this fashion generally *move* through the cellular space, depositing copies as they go, a design that has apparently never been adopted in past manually-created cellular automata models of replication. For example, the 4-component replicator in Figure 17 can be viewed as going through transformations as it translates to the right (relative to its initial position, which is marked by the origin of arbitrary coordinate axes in the figure), periodically reappearing in its original form (t=3,6,...) as it gives off replicants in the upper right quadrant (t=4,7,...) that themselves are rotated and moving upwards. Further details can be found in [37].

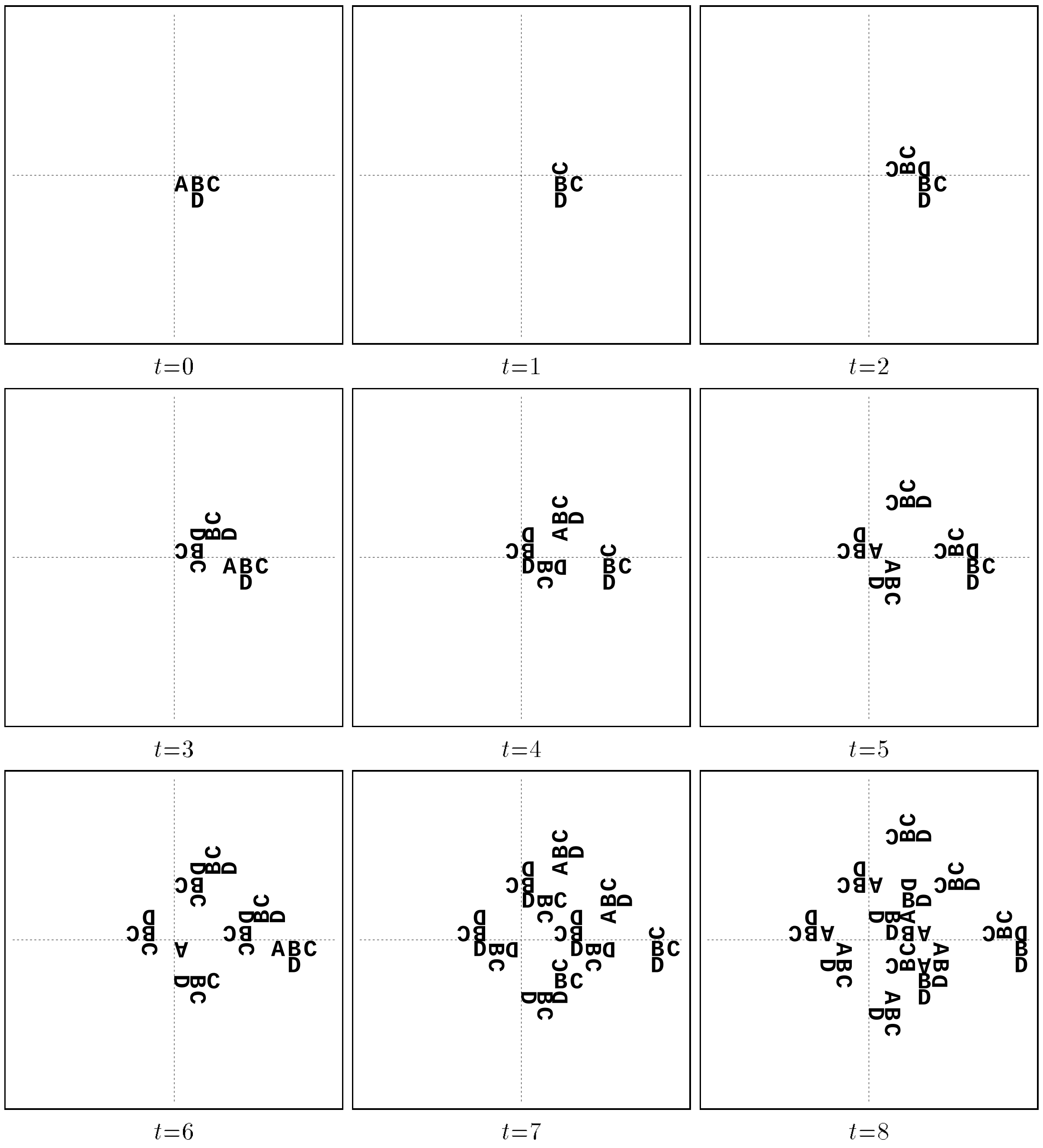


Figure 17: A 4-component self-replicating polyominoe. Its initial state is shown at the upper left (t=0). Several replicants can be seen by t=7 and t=8.

5. PROGRAMMING SELF-REPLICATING LOOPS

We observed earlier that, in discussing self-replicating systems in cellular automata, there are two levels of abstract machines: the individual cells of the cellular space, and the configuration of "components" that, as an aggregate, jointly represent a self-replicating structure. We can thus speak of programming either type of machine. In the former case, the rule table representing the transition function is the "program" directing a cell's behavior. In the latter case, the sequence of signals on a tape or loop that direct the self-replication forms a program. In this section we are solely concerned with the latter case when we refer to programming a self-replicating structure.

The concept of programming self-replicators can be traced back to von Neumann's original universal computer-constructor [70]. The set of instructions (signals) or description on the replicating structure's tape that describe its own structure can be viewed as the machine's program. Similarly, the sequence of instructions that circulate around a self-replicating loop form a program that directs the loop's replication. Such programs have only been concerned with replication of the loop in the past. During recent years, however, the idea of programming self-replicators to do more than just replicate has been receiving increasing attention. The underlying idea is that the signal sequences directing a structure's replication can be extended in some fashion to solve a specific class of problems while replication occurs. The motivation for such *programmed replicators* is that they provide a novel, massively parallel computational environment that may lead over the long term to powerful, very fast computing methods. Two different approaches have been taken so far.

5.1 Duplicated Program Sequences

Perhaps the most straightforward approach to programming self-replicating loops to solve problems is simply to extend the sequence of signals circulating around the loop, adding additional signals representing a program that carries out some task. This *application program* is copied along with the replication program unchanged from generation to generation as the loop replicates, and is executed once by each loop in between replications. The viability of this approach was recently demonstrated by programming partially sheathed loops to construct a pattern (the letters LSL) in the interior of each replicated loop [65]. Using a loop with four arms based on the 9-neighborhood, it was possible to create extra space on the loop for an application program by factoring more of the replication process into the loop's transition function. In other words, rather than the growth process of the child loop being directed to occur by a sequence of '>' signals as in the examples above, such growth/extension was the default that occurred automatically. The instruction sequence thus needed to consist only of appropriately delayed 'L' signals indicating when the growth process should change direction to start a new side of the loop. The price paid for automatic loop growth and the execution of an application program is in terms of the complexity of the rule set: typically, on the order of a few hundred rules are required in this situation [65].

```
a.     XXXXXXXX              b.     XXXXXXXX
      XO+ OL OLX                   XO+ OL OLX
      X XXXXXX X                   X XXXXXX X
      X+X    XOX                   X+X    XOX
      XOX    XOX                   XOX    XOX
      X X    XOX                   X X    XOX
      X+X    XOX                   X+X    XOX
      XOXXXXXXOXXXXX               XOXXXXXXOXXXXX
      X +O +O +OOOOOX              X +O +O +OOOOOX
       XXXXXXXXXXXXX               XAXXXXXXXXXXXX
                                   XPX    XDX
                                   XPX    XDX
                                   XPH    XDX
                                   XPX    XDX
                                   XPX    XDX
                                   XPX     X
                                   XPX
                                   XPX
                                    X
```

Figure 18: a. Self-replicating sheathed loop discussed earlier [33]; reproduced here for comparison purposes. b. Schematic illustration of how the sheathed loop can be extended to support programming [50]. Two potentially infinite, vertical "tapes" have been added, one to include a program (lower left) and one for data (lower right). Each P designates a simple program instruction and each D a unit of data. The P's and D's are replaced by specific signals in solving a problem.

A practical problem with the above approach to programming self-replicating loops is the restricted amount of space available along a loop for application programs and data. This problem can be solved by adding "tapes" to the loops [50]. This is analogous to the tapes used in the earliest universal computer-constructor replicators [11, 70]. The basic idea is illustrated in Figure 18. Starting with a sheathed loop (Fig. 18a), two vertically descending tapes of arbitrary size are added to one side of the loop (Fig. 18b). The left tape is used to store a signal sequence representing an application program (signal locations marked by P's), while the right tape is used to store problem data (item locations marked by D's). Reading heads, e.g., the H in the right part of the instruction tape sheath in Fig. 18, move up and down the tapes. As the program reading head reaches an instruction P, that instruction may cause a signal to move along the sheath to act on part of the data tape.

Using this approach with the 5-neighborhood, it has been shown that one can program a self-replicating loop to perform parentheses checking [50]. An expression with parentheses is represented on the data tape, and the program checks that the parentheses are well-formed or balanced, a computation that corresponds to recognition by a non-regular language. A parent loop first replicates itself and copies unchanged its program and data tapes onto its child loop. It then executes its program to balance parentheses. This process uses cells having 63 states and roughly 8500 state change rules in the transition function. Because of the presence of tapes, replication is restricted to the two horizontal directions only, at least in a two-dimensional cellular automata space.

It can be shown that tape-extended self-replicating loops are capable of executing any desired program [50]. Thus, in principle such extended self-replicating loops exhibit computational universality just as did the earliest self-replicating structures, yet they are qualitatively simpler.

5.2 Expanding Problem Solutions

The programmable self-replicating loops described above literally encode a set of instructions on the loop or an attached tape that directs solution of a problem. This application program is copied unchanged from parent to child, so each generation of loops is executing exactly the same program on exactly the same data. This demonstrates the feasability of programming self-replicating loops and might find use in some applications, but a more general approach would allow the program and data to change over time in some systematic fashion. While such a generalized approach has not been examined yet, it has proven possible to append potential problem solutions to the replication instruction sequence circulating on a self-replicating loop [9]. Unlike the above approach, the initial problem solution is not copied exactly from parent to child but is modified from generation to generation. Each child loop gets a different partial problem solution. If a loop determines it has found a valid complete problem solution, it stops replicating and retains that solution as a circulating pattern in its loop. On the other hand, if a loop determines its partial solution is not useful, the loop "dies", erasing itself without descendents. Thus, the process of forming a colony of loops can be viewed as a parallel state space search through the space of problem solutions. At the end of this process when replication has stopped, the cellular space contains one or more non-replicating loops, each with a circulating sequence of signals that encodes a valid problem-solution (assuming such a solution exists).

We recently applied this approach of generating possible solutions and selectively discarding non-viable ones to solve satisfiability problems (SAT problems), a classic example of an NP-complete problem [27]. Given a boolean predicate like $P = (\sim x_1 \vee x_3) \wedge (x_1 \vee \sim x_2) \wedge (x_2 \vee \sim x_3)$, the SAT problem is: "What assignment of boolean values to the binary variables x_1, x_2 and x_3 can satisfy this predicate?", i.e., what assignment can make this predicate evaluate to True? In this case, P will be true if $x_1 = 1, x_2 = 1$ and $x_3 = 1$, for example. The predicate P here is in conjunctive normal form, where each part of the predicate surrounded by parentheses is called a *clause*. A SAT problem is usually designated as an m-SAT problem if there are m boolean variables in a clause of its predicate. Therefore, the above example P is a 2-SAT problem.

Figure 19 illustrates the generate-and-select process for a self-replicating loop carrying 3 binary bits representing the three variables x_1, x_2 and x_3 used in predicate P. In the initial loop at $t = 0$, unexplored bits are represented by the symbol A. These A's replace some of the o's forming the data path in the self-replicating loop. The original growth signal '>' is also replaced by the symbol + reflecting some minor differences in the replication process (the data path symbol O used in earlier figures has also been changed here to lower case o typographically to avoid confusion with the digit zero). Explored bits are represented by either digit 0 ("false") or 1 ("true") in the loops. The bit sequence that a loop carries is read off counterclockwise starting right after the L symbol. Thus, for example, the lower left loop

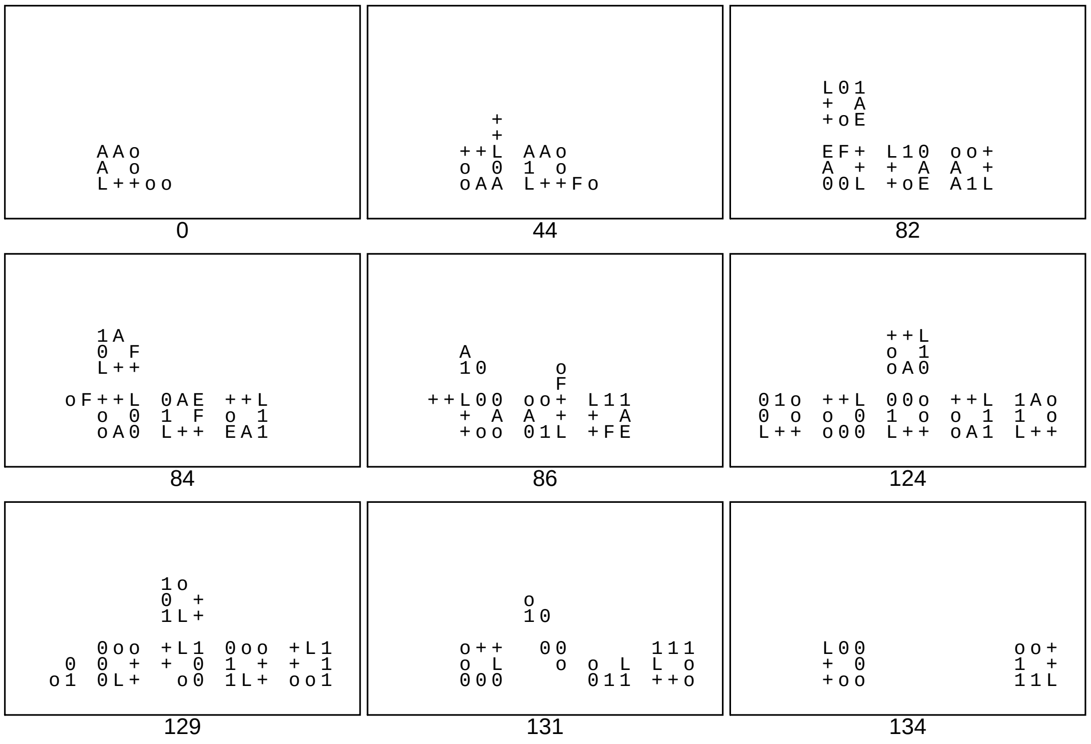


Figure 19: The generation and selection of satisfying boolean assignments by self-replicating loops for the predicate P given in the text. The monitoring of circulating loop signals by each cell provides the selection process. At time 0, the initial loop is placed in the cellular automata space, and carries unexplored binary bits represented as AAA. By time 44 the first replication cycle has completed and there are two loops in the cellular automata space. The first binary bit has been explored, resulting in the first A being converted into 0 and 1 in the two resulting loops. By time 82 the second replication cycle has completed and there are four loops in the cellular automata space. Starting at time 84 the top loop is being destroyed (note the missing corner cell of the loop). Its bit sequence '01A' does not satisfy the second clause in predicate $\mathcal{P}$, so it is being erased by the monitor underneath its top-right corner. At time 86 the erasing process continues while the other loops start their next replication cycle. At time 124 the third (also the last) replication stage is completed and there are six loops in the cellular automata space. Four of these loops do not survive the selection process for long and are erased (times 129 and 131). Finally, two satisfying assignments 000 and 111 remain in the cellular space at time 134.

in Fig. 19 at $t = 124$ carries the sequence 001.

Without the selection process, in three generations all eight possible boolean assignments for the variables used in P would appear, carried by eight loops in the cellular automata space, assuming that no collisions occurred. Loops stop replicating once they have explored all of their A bits. Since the exploration of bits is done one bit at a time at each generation, and since at each exploration step a different bit appears in the parent and child loops, we can be sure that all possible boolean assignments will be found with the generation process, if there are no collisions of loops in the space. If collisions do occur, a loop unable to replicate initially will continue trying until space appears for it to do so.

To remove those loops which do not satisfy a SAT predicate, each cell in the space serves as a *monitor*. Each monitor tests a particular clause of the SAT predicate. If the condition a cell is looking for in its role as a monitor is found, it will "destroy" the loop passing through it. For the specific predicate P, three classes of monitors, each testing for one of the following conditions, are planted in the cellular automata space: $x_1 \wedge \sim x_3$, $\sim x_1 \wedge x_2$, and $\sim x_2 \wedge x_3$. These conditions are just the negated clauses of P. If any one clause of predicate P is *not* satisfied, the whole predicate will not be satisfied. A monitor will destroy a loop passing through it if its corresponding clause is found to be unsatisfied by the bit sequence carried by the loop. This detection process is done in linear time since essentially each monitor is just a finite automata machine, and the bit sequence passing through it can be seen as the string for regular expression recognition. With enough properly distributed monitors in the cellular automata space, they can effectively remove all unsatisfying solutions.

Some steps of the generation and selection process for the same 3 x 3 loop are shown in Figure 19. Starting with one initial loop carrying a totally unexplored bit sequence AAA at $t = 0$, 0AA appears in the parent loop and 1AA in the child loop in the first generation ($t = 44$). In the second generation two new loops carrying 01A and 11A are obtained; the two parents now carry 00A and 10A. If all goes well, in the third and final generation, we should get four more loops 011, 111, 001 and 101; the four parents would carry 010, 110, 000 and 100. If no selection and no collisions occurred, then there should be all eight possible values for a 3 bit binary sequence. However, it can be seen in this figure that some of the loops are destroyed or never even generated after the second generation. For example, the topmost loop at $t = 82$ is erased ($t = 84$, $t = 86$). Since it has been found (by the monitors) that this loop's partially explored bits 01A do not satisfy one of the clauses, there is no need to explore further since all of its descendents will carry the same binary bits. In three generations only two loops are left in the cellular automata space instead of eight ($t = 134$). These two loops carry exactly the only two satisfying boolean assignments for the original SAT predicate P, which are 000 and 111. The details of this process are given in [9].

6. DISCUSSION

Cellular automata models of self-replication have been studied for almost fifty years now. In this article we have presented the view that work on this topic has involved at least three different approaches. The earliest work examined large, complex universal computer-constructors that are marginally realizable. This work established the feasibility of artificial self-replication, examined many important theoretical issues involved, and gradually examined progressively simpler self-replicating universal systems. A second and more recent approach has focussed on the design of self-replicating loops. Self-replicating loops are so small and simple that they have been readily realizable. Finally, we believe that a third approach merits investigation: the emergence of self-replicators from initially non-replicating systems. As examples of this, we discussed our recent studies of the emergence of self-replicating structures from randomly-distributed, non-replicating components, and the evolution of transition rules that support replication of small but arbitrary initial structures.

The recent work on self-replicating cellular automata models that is of most direct significance for computer science is that on programming self-replicating loops. As we have seen, problem-solving can be accomplished by self-replicating structures as they replicate. This can be achieved either by attaching a set of instructions (signals) to those directing replication, or by encoding a tentative problem-solution that systematically evolves into a final solution. Implementations have shown that programmed replicators are clearly capable of solving nontrivial problems. These programmed self-replicating structures are intriguing in part because they provide a novel approach to computation. This approach is characterized by massive parallelism (each cell in the underlying cellular automata space is simultaneously carrying out computation), and by the fact that both self-replication and problem-solving by replicators appears as emergent properties of solely local interactions.

While progress in creating and studying cellular automata models has accelerated during the last few years, a great deal remains to be done. A high level language that specifically supports development of cellular automata transition functions would be of great value to future investigations, as this is currently largely unavailable. Similarly, while hardware that directly supports the massively parallel but local computations of cellular automata modeling has appeared [23, 67], it is also largely unavailable today. If such software and hardware environments could be made available in the future, it would greatly reduce the large programming and processing times associated with research in this area.

Among the many issues that might be examined in the future, several appear to be of particular importance. These include the further development of programmable self-replicators for real applications, and a better theoretical understanding of the principles of self-replication in cellular automata spaces. More general and flexible cellular automata environments, such as those having non-uniform transition functions [61] or novel interpretations of transition functions [36], merit exploration. It has already proven possible, for example, to create simple self-replicating structures in which a cell can change the state of neighboring cells directly [36], or can copy its transition function into a neighbor cell while allowing cells to have different transition functions [61]. Also, from the perspective of realizing physically self-replicating devices, closer ties and exchange of information between the modeling work described here and ongoing work to develop self-replicating molecules and nanotechnology is important. Closely related to this issue is ongoing investigation of the feasibility of electronic hardware directly supporting self-replication [39, 40]. If these developments occur and progress, we foresee a bright and productive future for the development of a technology of self-replicating systems.

Finally, we expect that as the modeling of self-replication progresses, it will assume increasing importance in theoretical biology. Artificial self-replicators have already shown that self-replication of information-carrying structures can be far simpler than many people have realized [57]. Analogous conclusions about unexpectedly simple information processing requirements have been reached regarding other complex physical/chemical processes after cellular automata models of them were developed, such as the appearance of stably rotating spiral forms in the Belousov-Zhabotinskii autocatalytic reaction [18, 38]. Further, it seems probable that the simple self-replicating structures described here are not the only ones possible. The self-replicating structures discovered using a genetic algorithm (Figure 17) suggest that novel approaches still remain to be identified.

At present it has not been possible to actually realize any "informational replicating systems" in the biochemistry laboratory [46], although recent results in experimental chemistry suggest this may someday be possible [1, 15, 26, 69]. The replicating loops and polyominoes described here provide intriguing ideas for self-replicating molecular systems, but are not intended as realistic models of known biochemical processes and have only a vague correspondence to real molecular structures. The information-carrying loop, for example, might be loosely correlated, with a circular oligonucleotide, and the construction arm with a protein that reads the encoded replication algorithm to create the replica. Still, the existence of these systems raises the question of whether contemporary techniques being developed by organic chemists studying autocatalytic systems or the innovative manufacturing techniques currently being developed in the field of nanotechnology could be used to realize self-replicating molecular structures patterned after the information processing occurring in simple self-replicating cellular automata structures.

References

[1] I. Amato, Capturing Chemical Evolution in a Jar, *Science*, 255, 1992, 800.

[2] M. Arbib, Simple Self-Reproducing Universal Automata, *Inform. and Control*, 9, 1966, 177-180.

[3] E. Banks, Universality in Cellular Automata, *Eleventh Ann. Symp. on Switching and Automata Theory*, IEEE, 1970, 194-215.

[4] E. Berlekamp, J. Conway and R. Guy, *Winning Ways for Your Mathematical Plays*, (Academic Press, New York, 1982), vol. 2, chap. 25.

[5] A. Burks, in *Essays on Cellular Automata*, A. Burks, Ed. (University of Illinois Press, Urbana, 1970), chap. 1.

[6] J. Byl, Self-Reproduction in Small Cellular Automata, *Physica D*, **34**, 1989, 295-299.

[7] J. Case, Periodicity in Generations of Automata, *Mathematical Systems Theory*, 8, 1974, 15-32.

[8] H. Chou & J. Reggia, Emergence of Self-Replicating Structures in a Cellular Automata Space, *Physica D*, 1997, in press.

[9] H. Chou & J. Reggia Solving SAT Programs with Self-Replicating Loops. In preparation, 1997.

[10] H. Chou, J. Reggia, R. Navarro-Gonzalez, & J. Wu. An Extended Cellular Space Method for Simulating Autocatalytic Oligonucleotides, *Computers and Chemistry*, 18, 1994, 33-43.

[11] E. Codd, *Cellular Automata*, Academic Press, 1968.

[12] J. Demongeot, E. Goles & M. Tchuente (eds.) *Dynamical Systems and Cellular Automata*, Academic Press, 1995.

[13] K. Drexler, Biological and Nanomechanical Systems, in *Artificial Life*, C. Langton, Ed. (Addison-Wesley, New York, 1989), 501-509.

[14] D. Farmer, T. Toffoli & S. Wolffram (eds.), *Cellular Automata*, North Holland, 1984.

[15] Q. Feng, T. Park and J. Rebek, Crossover Reactions Between Synthetic Replicators Yield Active and Inactive Recombinants, *Science*, **256**, 1992, 1179-1180.

[16] R. Freitas & W. Gilbreath (eds.) *Advanced Automation for Space Missions*, NASA Conference, Publication 2255, NTIS, 1982.

[17] M. Gardner, The Fantastic Combinations of John Conway's New Solitaire Game "Life", *Scientific American*, 223(4):120-123, 1970.

[18] M. Gerhardt, Schuster H & Tyson J. A Cellular Automaton Model of Excitable Media Including Curvature and Dispersion, *Science*, 247, 1990, 1563-1566.

[19] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

[20] J. Grefenstette, Genetic Algorithms and Their Applications, in *Encyclopedia of Computer Science and Technology*, vol. 21, Suppl. 6, Marcel Dekker, 1990, 139-152.

[21] H. Gutowitz (eds.), *Cellular Automata-Theory and Practice*, MIT Press, 1991.

[22] G. Herman, On Universal Computer-Constructors, *Information Processing Letters*, **2**, 1973, 61-64.

[23] W. Hillis *The Connection Machine*, MIT Press, 1985.

[24] J. Holland *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[25] J. H. Holland, Studies of the Spontaneous Emergence of Self-Replicating Systems Using Cellular Automata and Formal Grammars. In *Automata, Languages, Development,* A. Lindenmayer and G. Rozenberg (eds), North-Holland, pp. 385–404, 1976.

[26] J. Hong, Q. Feng, V. Rotello and J. Rebek, Competition, Cooperation and Mutation: Improving a Synthetic Replicator by Light Irradiation, *Science*, **255**, 1992, 848-850.

[27] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Language and Computation*, (Addison-Wesley, Reading, 1979), Chap. 7.

[28] J. Ibanez, D. Anabitarte, et al. Self-Inspection Based Reproduction in Cellular Automata, in *Proc. 3rd Euro. Conf. Artif. Life*, F. Moran et al, eds., Springer, 1995, 564-576.

[29] H. Jacobson, *Amer. Sci.*, On Models of Self-Replication, *Amer. Sci.* **46**, 1958, 255-284.

[30] J. Kephart, A Biologically Inspired Immune System for Computers, in R. Brooks & P. Maes (eds.). *Artificial Life IV*, MIT Press, 1994, 130-139.

[31] J. Koza, *Genetic Programming*, MIT Press, 1992.

[32] R. Laing, Some Alternative Reproductive Strategies in Artificial Molecular Machines, *J. Theor. Biol.,* **54**, 1975, 63-84.

[33] C. Langton, Self-Reproduction in Cellular Automata, *Physica 10D*, 1984, 135-144.

[34] C. Langton, Ed., *Artificial Life* (Addison-Wesley, New York, 1989);

[35] C. Langton, C. Taylor, J. Farmer and S. Rasmussen, Eds., *Artificial Life II* (Addison-Wesley, New York, 1992).

[36] J. Lohn and J. Reggia, Discovery of Self-Replicating Structures using a Genetic Algorithm, *IEEE Internat. Conf. on Evolutionary Computing*, Perth, 678–683, 1995.

[37] J. Lohn and J. Reggia. Automatic Discovery of Self-Replicating Structures in Cellular Automata, in preparation, 1997.

[38] B. Madore and W. Freedman, Computer Simulations of the Belousov-Zhabotinsky Reaction, *Science*, **222**, 1983, 615-616.

[39] D. Mange, M. Goeke, D. Madon, et al. Embryonics: A New Family of Coarse-Grained Field-Programmable Gate Array with Self-Repair and Self-Reproducing Properties, *Towards Evolvable Hardware*, Springer Verlag, 1996, 197-220.

[40] D.Mange, A. Stauffer, & G. Tempesti. Self-replicating and Self-repairing Field-Programmable Processor Arrays with Universal Construction/Computation, *Proc. 15th Internat. Joint Conf. on Artif. Intell.*, Workshop on Evolvable Systems, Morgan Kaufmann, 1997, in press.

[41] R. Merkle Self-Replicating Systems and Low Cost Manufacturing, in M. Welland & J. Gimzewski (eds), *The Ultimate Limits of Fabrication and Measurement*, Kluwer, 1994, 25-32.

[42] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, 1996.

[43] E. Moore, Machine Models of Self-Reproduction, *Proc. Fourteenth Symp. Appl. Math*, American Mathematical Society, 1962, 17-33.

[44] H. Morowitz, A Model of Reproduction, *Amer. Sci.*, **47**, 1959, 261-263.

[45] J. Myhill, The Abstract Theory of Self-Reproduction, in *Views on General Systems Theory*, M. Mesarović, Ed. Wiley, 1964, 106-118.

[46] L. Orgel, Molecular Replication, *Nature*, **358**, 203-209 (1992).

[47] J. Oró, S. Miller and A. Lazcano. The Origin and Early Evolution of Life on Earth, *Ann. Rev. Earth Planet. Sci.*, **18**, 1990, 317-356.

[48] A. Pargellis, The Evolution of Self-Replicating Computer Organisms, *Physica D*, 98, 1996, 111-127.

[49] L. Penrose, Mechanics of Self-Reproduction, *Ann. Human Genetics*, 23, 1958, 59-72.

[50] J. Perrier, M. Sipper & J. Zahnd. Toward a Viable, Self-Reproducing Universal Computer, *Physica D*, 97, 1996, 335-352.

[51] U. Pesavento, An Implementation of von Neumann's Self-Reproducing Machine, *Artificial Life*, 2 (4), pp. 337–354, 1995.

[52] C. Ponnamperuma, Y. Honda, and R. Navarro-González, Chemical Studies on the Existence of Extraterrestrial Life, *J. Brit. Interplanet. Soc.*, **45**, 1992, 241-249.

[53] K. Preston and Duff M. *Modern Cellular Automata*, Plenum, 1984.

[54] L. Priese, On a Simple Combinatorial Structure for Syblying Nontrivial Self-Reproduction, *J. Cybernet*, **6**, 1976, 101-137.

[55] S. Rasmussen, Knudsen C., Feldberg R., & Hindsholm M. The Coreworld: Emergence and Evolution of Cooperative Structures in a Computational Chemistry, *Physica D*, 42, 1990, 111-134.

[56] T. Ray. Evolution, Ecology and the Optimization of Digital Organisms, *Santa Fe Working Paper 92-08-042*, 1992.

[57] J. Reggia, S. Armentrout, H. Chou, and Y. Peng. Simple Systems That Exhibit Self-Directed Replication, *Science*, 259:1282-1288, 1993.

[58] J. Reggia, H. Chou, S. Armentrout & Y. Peng, Transition Functions and Software Documentation, Technical Report, CS-TR-2965, Dept. of Computer Science, UMCP, 1992.

[59] F. Richards, Meyer T & Packwood N. Extracting Cellular Automata Rules Directly from Experimental Data, *Physica D*, 45, 1990, 189-202.

[60] R. Rosen, On a Logical Paradox Implicit in the Notion of a Self-Reproducing Automaton, *Bull. Math. Biophys.*, **21**, 1959, 387-394.

[61] M. Sipper, Studying Artificial Life Using a Simple, General Cellular Model, *Artificial Life*, 2 (1), pp. 1–35, 1995.

[62] M. Sipper & E. Ruppin, Co-Evolving Architectures for Cellular Machines, *Physica D*, 99, 1997, 428-441.

[63] A. Smith, in *Artificial Life II*, C. Langton, C. Taylor, J. Farmer and S. Rasmusen, Eds., Addison-Wesley, 1992, 709.

[64] P. Tamayo and Hartman H. Cellular Automata, Reaction-Diffusion Systems, and the Origin of Life, in *Artificial Life*, C. Langton (editor), Addison-Wesley, 1989.

[65] G. Tempesti, A New Self-Reproducing Cellular Automaton Capable of Construction and Computation. In *Proc. Third European Conference on Artificial Life,* F. Moran, A. Moreno, J. Morelo, and P. Chacon (eds), Springer, 555–563, 1995.

[66] J. Thatcher, Universality in the von Neumann Cellular Model, in *Essays on Cellular Automata*, A. Burks, Ed., Univ. of Illinois Press, Urbana, 1970, 132-186.

[67] T. Toffoli and Margolus N. *Cellular Automata Machines*, MIT Press, 1987.

[68] P. Vitányi, Sexually Reproducing Cellular Automata, *Math. Biosci.*, **18**, 23 (1973).

[69] G. von Kiedrowski, A Self-Replicating Hexadeoxynucleotide, *Angew. Chem. Int. Ed. Engl.*, **25**, 1986, 932.

[70] J. von Neumann *The Theory of Self-Reproducing Automata*, University of Illinois Press, Urbana 1966.

[71] K. Williams, Simplifications of a Self-Replication Model, *Science*, 261, 1993, 925.

[72] S. Wolfram, *Theory and Applications of Cellular Automata*, World Scientific, 1986.

[73] S. Wolfram, *Cellular Automata and Complexity*, Addison Wesley, 1994.