J.A. Reggia, J.D. Lohn, H.H. Chou, ``Self-replicating Structures: Evolution, Emergence, and Computation,'' Artificial Life, vol. 4, no.3, 1998, pp. 283-302.

# Self-Replicating Structures: Evolution, Emergence and Computation

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

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

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

**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 was 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. We view past research 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 recent studies in the latter category 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.

# 1. MODELING SELF-REPLICATION

Computational modeling of self-replicating structures/machines has often been based on cellular automata. We view these cellular automata models as primarily taking three approaches. First, early self-replicating structures (1960's and 1970's) were large, complex universal systems modeled after Turing machines. They provided the first demonstration that artificial self-replicating structures could in principle be devised and stimulated substantial theoretical work. A second generation of self-replicating structures, self-replicting loops 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. More recently, we and others have taken a third approach 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.

The models of self-replication considered below are implemented in a cellular automata framework [8, 10, 32]. With any cellular automata model each cell's state transitions are governed by a set of rules forming the transition function. Each transition 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 transition rules, 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.

The mathematician John von Neumann first used cellular automata to study the logical organization of self-replicating structures [31]. 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. 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 (von Neumann neighborhood), and is literally a simulated digital computer (Turing Machine) that used a "construction arm" in a step-by-step fashion to construct a copy of itself from instructions on a "tape". 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 [3]. 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. 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 design of von Neumann's original universal computer-constructor can be found in [3, 31].

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 [3, 31]. Much subsequent work focused on finding simpler self-replicating structures. For example, investigators showed that some simplification of von Neumann's configuration was possible by redesigning specific components [29] or by increasing cell state complexity [2]. Most influential among this early work was Codd's demonstration that if the components or cell states meet certain symmetry requirements, then von Neumann's model could be done in a simpler fashion using cells having only eight states rather than the 29 used originally [7]. Codd argued that using components that were symmetrical led to a simpler model, and he created a universal computer-constructor that was simpler but otherwise similar in spirit to

that of von Neumann's. Another approach taken to reducing the complexity of von Neumann's design in a 2D cellular space focused on using more complex components [2].

## 2. SELF-REPLICATING LOOPS

While these early studies describe structures that self-replicate, the structures involved 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 been implemented [24]. The complexity of these early cellular automata models seems consistent with the remarkable complexity of biological self-replicating systems: they appear to suggest that self-replication is an inherently complex phenomenon. More recent work with self-replicating loops provides evidence that this is not necessarily so.

A much simpler self-replicating structure based on 8-state cells, which we refer to as a self-replicating sheathed loop, was developed by Langton in the mid-1980's (see Fig. 1b) [14]. The term "sheathed" here indicates that this structure is surrounded by a covering or sheath (X's in Fig. 1a-c). Consider Fig. 1a 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 data path forming the loop. 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. Non-replicating loops like this served as storage elements in the universal computer-constructors designed by von Neumann and Codd.

Fig. 1b shows the initial state of a self-replicating sheathed loop [14]. 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. 1a (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. The instruction sequence is used both as instructions that are

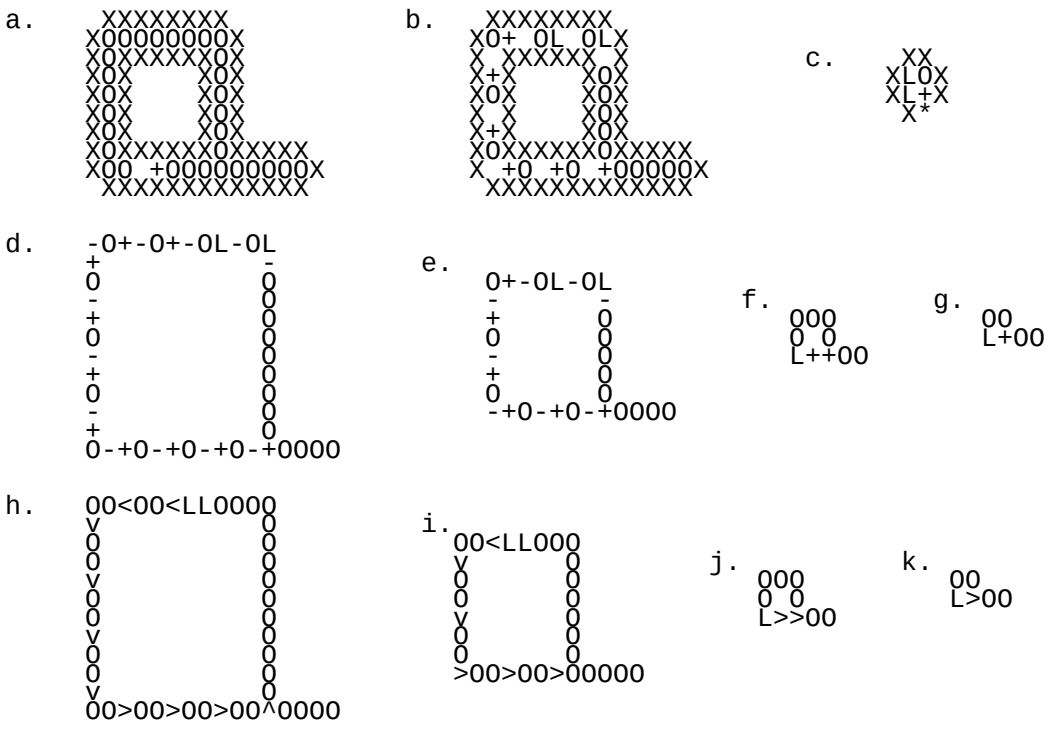


Figure 1: Self-replicating loops in two dimensional cellular automata. Cells in the quiescent state are indicated by blank spaces. (a) Sheathed but non-replicating loop. A core of O's is surrounded by a sheath of X's. A single signal (+ followed by blank space) repeatedly circulates counterclockwise around the loop. (b) A self-replicating sheathed loop; (c) A small self-replicating sheathed loop[4]; (d)-(g) Unsheathed self-replicating loops; (h)-(k) Unsheathed self-replicating loops.

interpreted to direct the construction of a replica, and as uninterpreted data that is copied onto the replica [14]. Thus, self-replicating loops are truely "information replicating systems" in the sense that this term is used by organic chemists [21].

The "program" of the replicating sheathed loop, pictured in Fig. 1b, 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.

We hypothesized that sheathed loops could be modified to produce even simpler and smaller self-replicating structures [25]. An unsheathed version of the original sheathed loop is shown in Fig. 1e. The signal sequence +–+–+–+–+–L–L– directing self-replication of this unsheathed loop is the exact same program as that of the sheathed loop, but written using different "instruction codes" ($+-$ for "extend", $L-$ for "extend and turn left"). As illustrated in Fig. 2, 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. 2b-d). As shown in Fig. 2e, after 150 iterations or units of time the original structure (on the left, its construction arm having moved to the top) has created a duplicate of itself (on the right). After several generations a single initial unsheathed loop results in an expanding "colony" where actively replicating structures are found only around the periphery.

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 in Fig. 1d-g. 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 [7]. The smallest unsheathed loop in this specific group (Fig. 1g) is more than an order of magnitude smaller than the original sheathed loop and requires only 174 transition rules.

In the past, 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 [3, 29, 31]. These oriented cell states were such that they permuted among one another consistently under successive 90° rotations of the underlying two-dimensional coordinate system. For example, the cell state designated

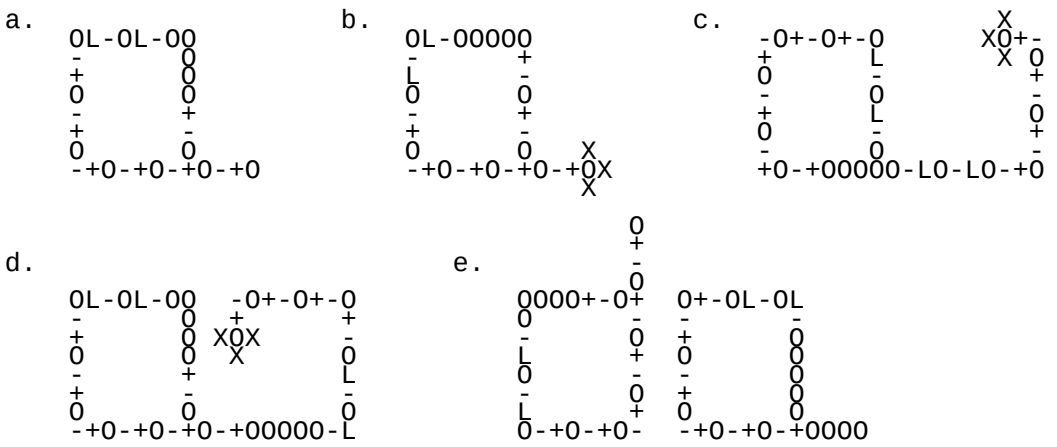


Figure 2: Successive states of a self-replicating unsheathed loop starting at time t=0. The instruction sequence repeatedly circulates counterclockwise around the loop with a copy periodically passing onto the construction arm. At t=3 (**a**) the sequence of instructions has circulated 3 positions counterclockwise 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.

$\uparrow$ 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 [7] and the simpler replicating loops ([14] and Fig. 1d-g) 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.

A second family of self-replicating unsheathed loops was developed, as shown in Fig. 1h-k, whose initial state and instruction sequence are similar to those already described in Fig. 1d-g. However, for the structures in Fig. 1h-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

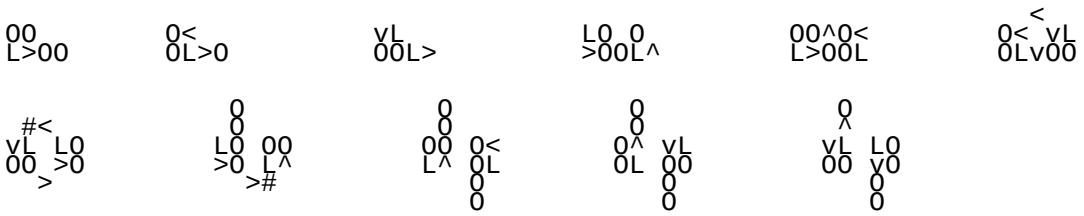

Figure 3: This small self-replicating loop uses only five unique components. Shown here are eleven immediately successive structures ordered left to right, top to bottom. Starting at $t = 0$, the initial state shown at the upper left passes through a sequence of steps until at $t = 10$ (last structure shown) an identical but rotated replica has been created.

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 (. O # L) are unoriented. For example, in Fig. 1i 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. 1a-g), they also can be viewed as simpler in that they have only 5 distinct possible components. Relaxing the strong rotational symmetry requirement like this consistently led to transition functions requiring fewer rules than the corresponding strong symmetry version [25]. This simplicity and speed of replication made possible by weak rotational symmetry are illustrated in Fig. 3 where the complete first replication cycle of a small unsheathed loop is shown. Only 31 rules are needed to direct replication.

## 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 different 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.

### 4.1. Emergence of Replicators

Recent work by our group has shown that it is possible to create cellular automata models in which self-replicating loops emerge from an initial state having a random density and distribution of components (the "primordial soup") [5]. These emergent self-replicating loops employ a general purpose rule set that supports replication of loops of different sizes and their growth. 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 ($40 \times 40$) cellular automata space using an initial component density of 25% is shown in Figure 4. 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 4), 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.

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 4), very small self-replicating loops of size $2 \times 2$ and $3 \times 3$ 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 ($40 \times 40$ 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 repli-

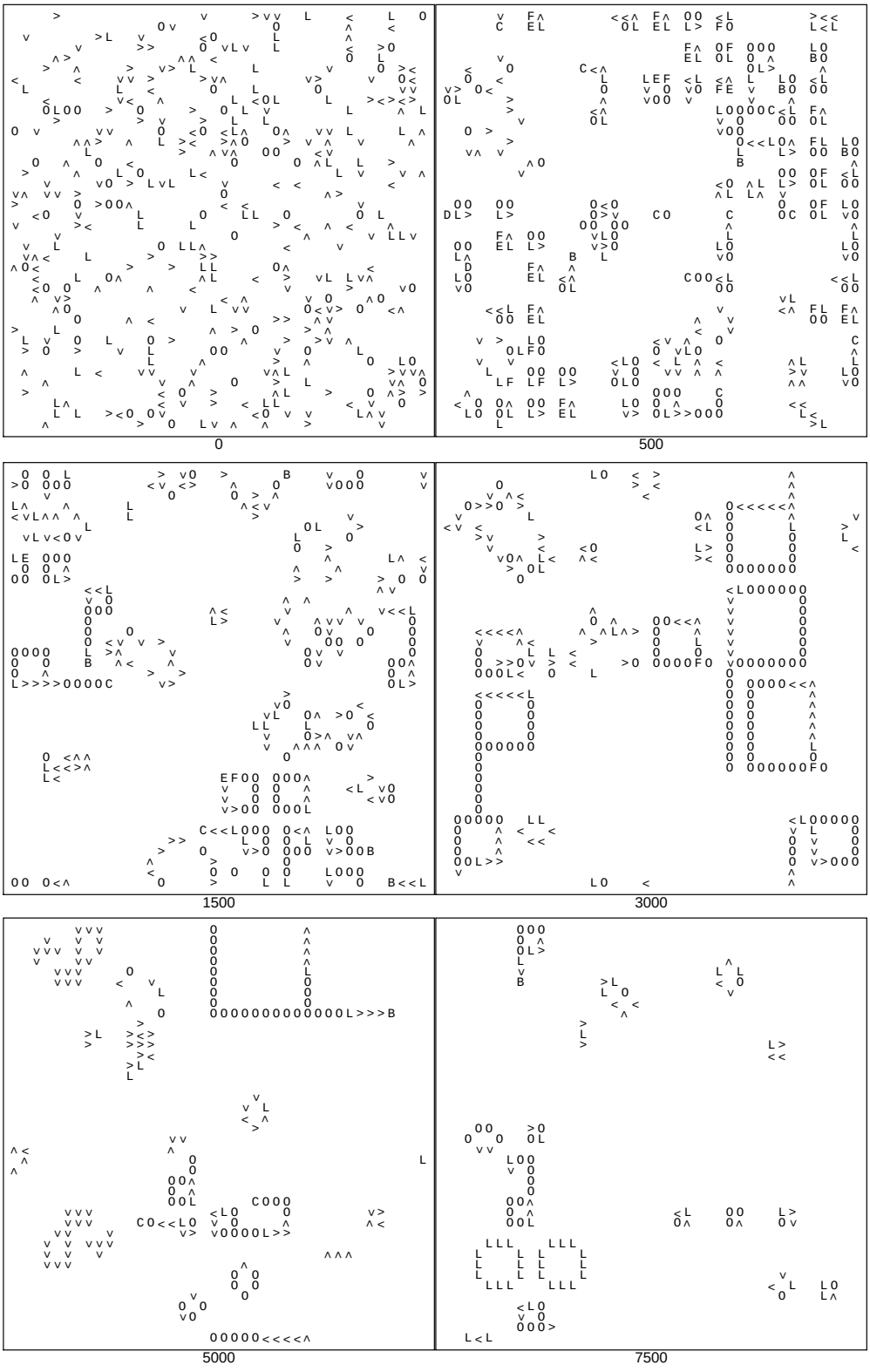


Figure 4: A running example of emergent self-replication. Times are shown.

cants 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. As is increasingly being done in cellular automata modeling, the transition function is based on a functional division of data fields [30]. As seen in Figure 5, 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.

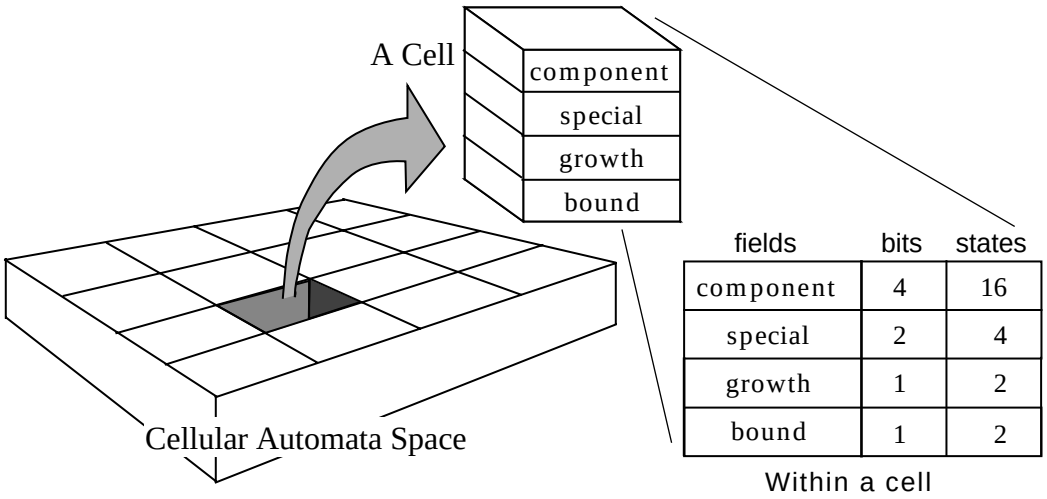


Figure 5: The 8 bit state variable in each cell is conceptually sliced into four different bit groups called *fields*. Each field represents a specific piece of information.

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 5).

The four data fields (Figure 5) 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 [14, 25]. 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. 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 6, this is a two step strategy. First, if a signal > finds 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 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:

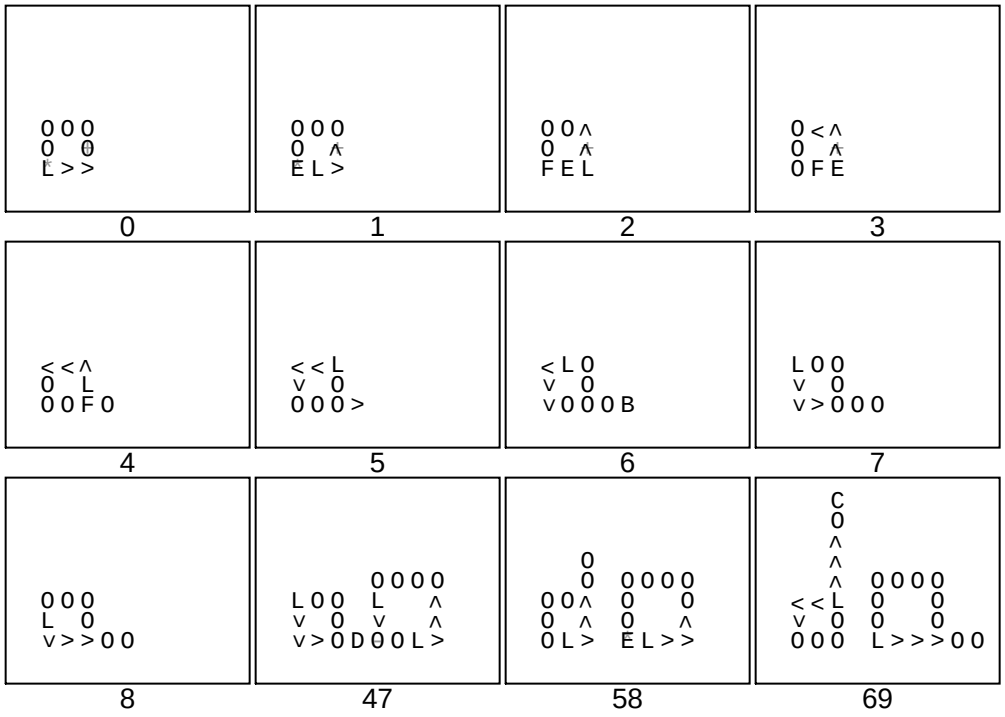


Figure 6: The growth of a larger loop (extended replication). At time 0 the branch special flag in the lower left cell and the growth bit in the middle right cell are both set. At time 2 the normal arm branching EF signal sequence is generated. At time 3 the signal sequence becomes >>> and subsequently 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, and 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

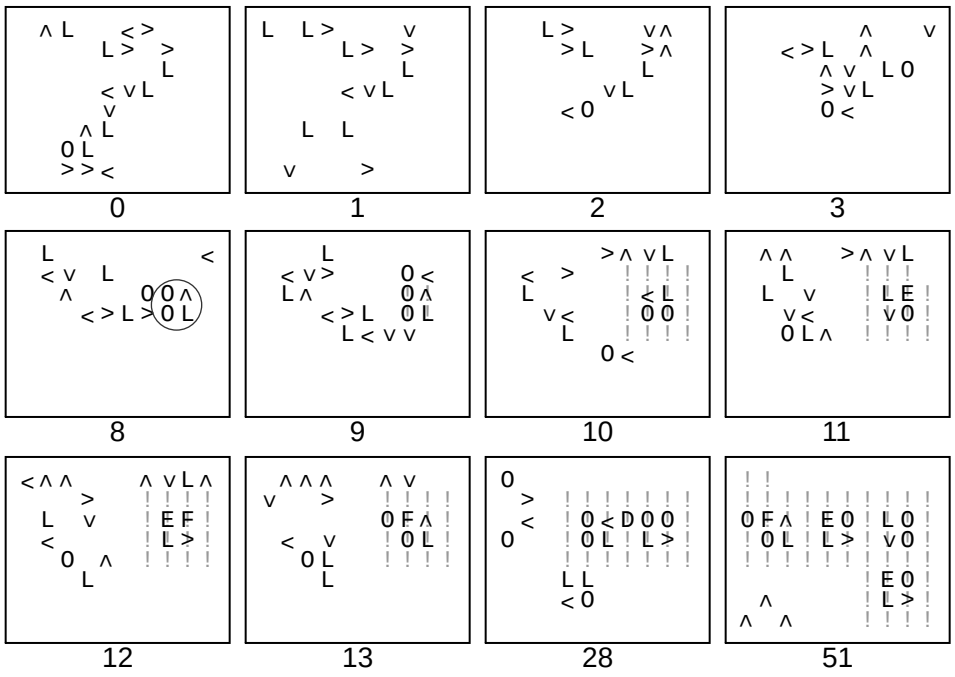


Figure 7: The emergence of a self-replicating structure. Components of structures are marked by a non-zero bound bit, or an '!' mark. At time 0 a randomly generated initial space is given. This space has only unbound components until time 8, when the pattern of the smallest replicating loop (circled) 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.

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 7.

The behavior of this model of emerging self-replication has been examined experimentally [5]. 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

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 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.

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 [13], and [22]), but these have not used cellular automata methods. For example, the investigation in [22] 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 Evolving Replicator Rules

Previous computational models of self-replication using cellular automata have been manually designed, a difficult and time-consuming process that is prone to the subjective biases of the implementer. As an alternative, we have recently shown that it is possible to automatically discover rules for self-replication using genetic algorithms [15, 16]. 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.

Relatively few previous studies have reported using genetic algorithms (or related techniques) to automatically produce rule tables for cellular automata (see, for example, [1, 20, 26]). With the exception of our preliminary report [15], there are no past reports of using GAs to discover self-replicating structures in cellular space models. Such research has most likely not been undertaken for at least two reasons. First, the computational load can become enormous. Rule tables for modest systems can quickly grow extremely large (e.g., 25,000 transition rules for a ten-state, five-neighbor, strongly rotation symmetric model), and manipulating numerous large rule tables with a genetic algorithm is very computationally expensive. Second, identification of an effective fitness functions is a difficult task. Apparently obvious fitness function, such as those that simply count the number of replicants, are useless early on as there will typically be no replicants. Further, comparing a developing structure to a predefined replicant template by way of pattern matches fails to give partial credit during the replication cycle itself, when the structure has changed its configuration as it undergoes replication. In other words, it is not obvious in advance at which time steps the quality of self-replication should be decided. Using cellular space state data from a single time step would require knowing *a priori* in which configuration will replicants appear and assumes that replicants appear all at once rather than at different time steps.

Fortunately, it has proven possible to solve these problems, at least to a limited extent [15, 16]. The genetic algorithm we used 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

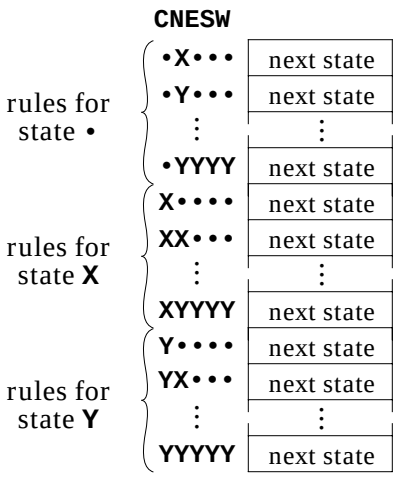


Figure 8: Encoding of a rule table used to represent a chromosome.

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 combined through crossover, and randomly altered by mutation. 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 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 8 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. 8). For the simulations described below, chromosomes were roughly 850

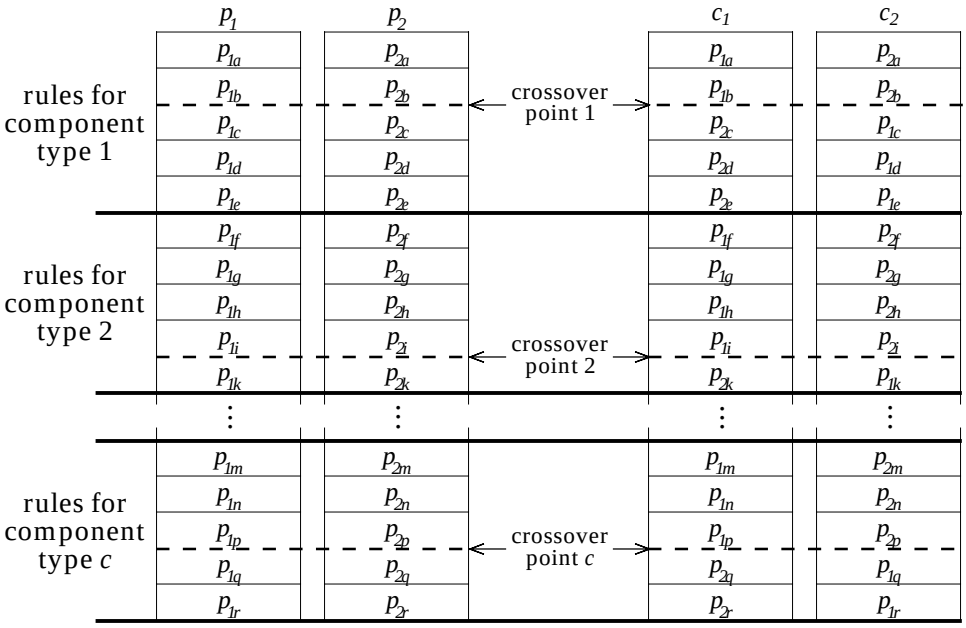


Figure 9: Illustration of crossover using rule tables. Parents $p_1$ and $p_2$ (left) are recombined to form offspring $c_1$ and $c_2$ (right) by segmenting the rule tables into $c$ partitions according to component type, and crossing over transition rules within each partition, with each crossover point chosen at random.

next-state elements long.

The type of crossover used here was a version of multi-point crossover whereby single-point crossover is applied within segments marked by the heavy lines in Fig. 9 (actual component types had many more transition rules than are shown here). A crossover point was randomly selected within each segment, and single-point crossover occurred in each segment. Empirical results comparing this crossover technique to that of single-point crossover (across the entire rule table) showed better performance for the multiple application of crossovers. After selection and crossover, each transition rule was subject to mutation which occurred by randomly choosing a new state.

Creating a fitness function $F$ that accurately measures the promise of a rule table for 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 these measures 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 [16]. 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 10. This self-replicating polyomino is 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 by the genetic algorithm in this fashion 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, these replicators 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 10 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. The fact that the self-replicating structures discovered using a genetic algorithm differ in unexpected ways from past designed replicants suggests that further exploration in the space of possible self-replicating structures using genetic algorithms will yield additional new structures.

## 5. PROGRAMMING SELF-REPLICATING LOOPS

The concept of programming self-replicators can be traced back to von Neumann's original universal computer-constructor [31]. 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-replicating loops 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.

One approach to programming self-replicating loops to solve problems is to extend the

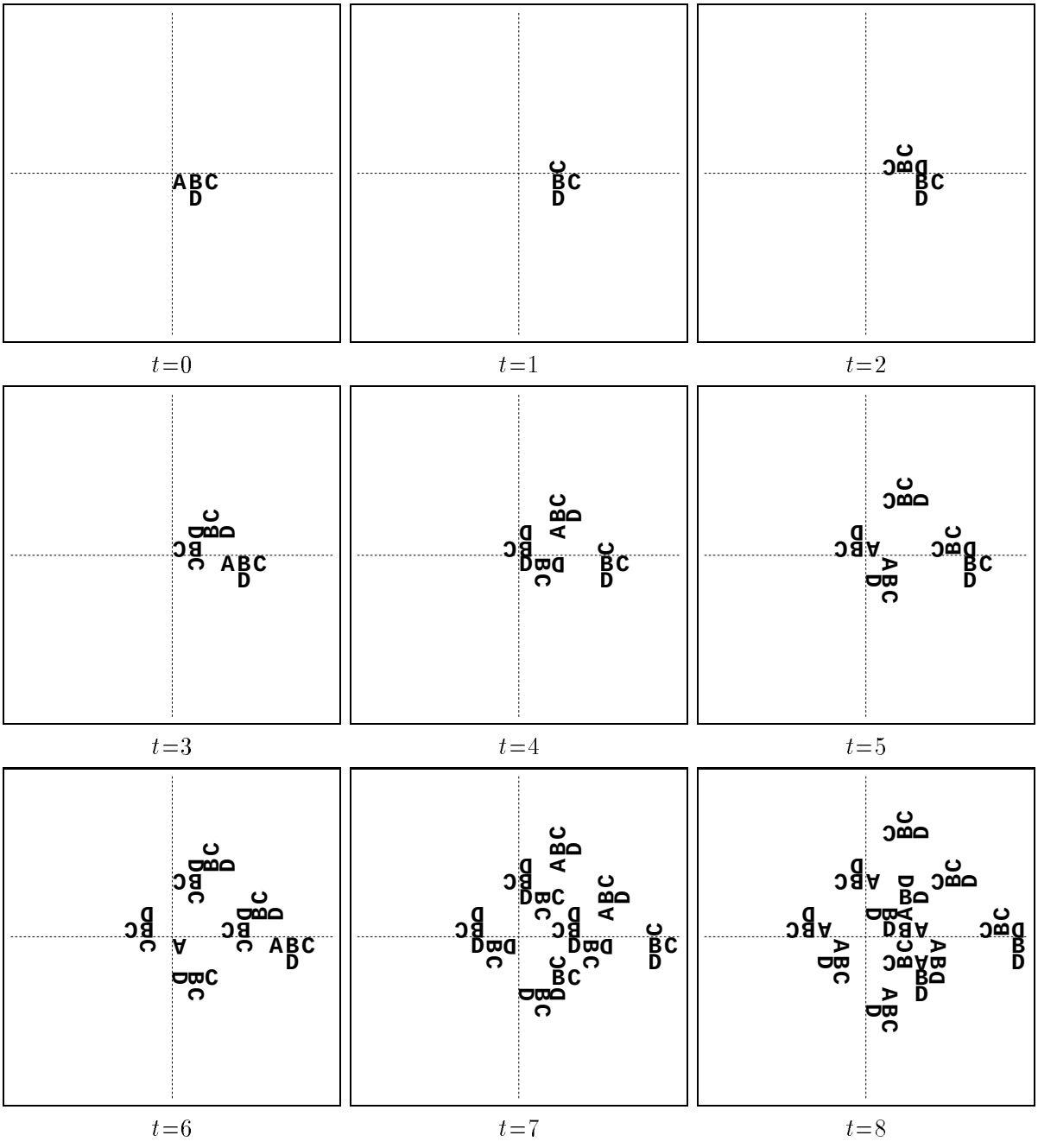


Figure 10: A 4-component self-replicating structure. Its initial state is shown at the upper left (t=0). The seed structure moves towards the right on successive time steps and produces two replicants: the first is seen at $t$=4 and then again at $t$=7 along with the second replicant (upper right quadrant of each respective frame). These replicants are rotated 90° counterclockwise and proceed upward. During the production of the first replicant, debris forms (near coordinate system origin of $t$=3 and $t$=4) and coalesces into two structures seen at $t$=5, lower left. One structure moves downward and attempts to self-replicate but due to crowding, is unable. The other moves to the left and produces its first replicant at $t$=8 (lower left quadrant).

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 in the interior of each replicated loop [28]. 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. 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 [28].

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 [23]. This is analogous to the tapes used in the earliest universal computer-constructor replicators [7, 31]. With two tapes, one can be used to store a signal sequence representing an application program, while the second can be is used to store problem data. Using this approach with the 5-neighborhood, it has been shown that one can program a self-replicating loop to perform parentheses checking [23]. 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. This process uses cells having 63 states and roughly 8500 state change rules in the transition function. It can be shown that tape-extended self-replicating loops are capable of executing any desired program [23]. Thus, in principle such extended self-replicating loops exhibit computational universality just as did the earliest self-replicating structures, yet they are qualitatively simpler.

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. We have recently examined a different approach in which potential problem *solutions* are appended to the replication instruction

sequence circulating on a self-replicating loop [6]. Unlike past approaches, 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 [12]. 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 11 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 clockwise starting right after the L symbol. Thus, for example, the lower left loop in

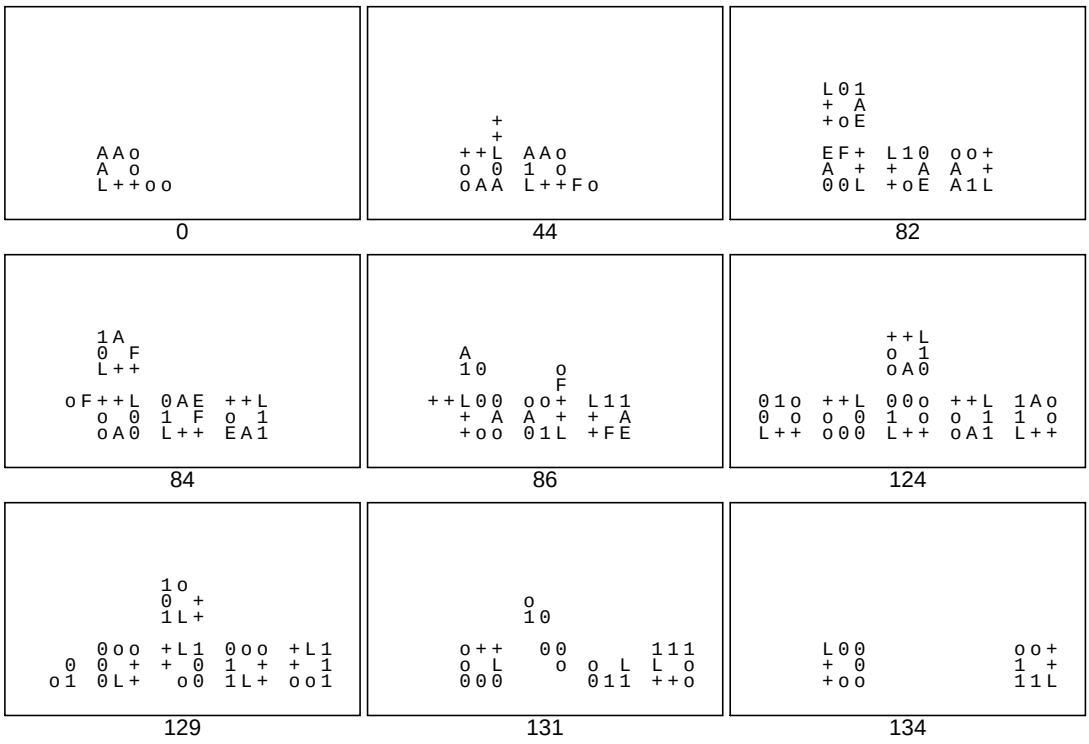


Figure 11: The generation and selection of satisfying boolean assignments by self-replicating loops for the predicate P given in the text. The monitoring of circulating loop signals by each cell provides the selection process. At time 0, the initial loop is placed in the cellular automata space, and 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.

Fig. 11 at $t = 124$ carries the sequence 001.

Without a 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 a bit sequence passing through it can be seen as a 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 11. 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.

## 6. DISCUSSION

Cellular automata models of self-replication have been studied for almost fifty years. 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.

Recent work has also shown that programmed replicators are capable of solving non-trivial 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 appear 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 [11, 30], 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[1].

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 [27] or novel interpretations of transition functions [15], 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 [15], or can copy its transition function into a neighbor cell while allowing cells to have different transition functions [27]. 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 [18, 19]. 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 [25]. Analogous conclusions about unexpectedly simple information processing requirements have been reached regarding other complex physical/chemical processes after cellular

[1]We are currently developing such a language called TREND. For information contact Dr. Chou at hhchou@tigr.org

automata models of them were developed, such as the appearance of stably rotating spiral forms in the Belousov-Zhabotinskii autocatalytic reaction [9, 17]. 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 suggest that novel approaches still remain to be identified.

# References

[1] D. Andre, F. Bennett, and J. Koza, Evolution of Intricate Long-distance Communication Signals in Cellular Automata using Genetic Programming, in *Artificial Life V*, C. Langton and K. Shimohara, Eds., Cambridge: MIT Press, 1997.

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

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

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

[5] H. Chou & J. Reggia, Emergence of Self-Replicating Structures in a Cellular Automata Space, *Physica D*, 110,1997, 252-272.

[6] H. Chou & J. Reggia, Problem Solving During Artificial Selection of Self-Replicating Loops, *Physica D*, 1998, in press.

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

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

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

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

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

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

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

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

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

[16] J. Lohn and J. Reggia. Automatic Discovery of Self-Replicating Structures in Cellular Automata, *IEEE Trans. on Evolutionary Computation*, 1, 1997, 165-178.

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

[18] 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.

[19] 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.

[20] M. Mitchell, P. Hraber, and J. Crutchfield, Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations, *Complex Systems*, 7, 1993, 89–130.

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

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

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

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

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

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

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

[28] 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.

[29] 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.

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

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

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