Physica 10D (1984) 135–144
North-Holland, Amsterdam

SELF-REPRODUCTION IN CELLULAR AUTOMATA

Christopher G. LANGTON
Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor, Michigan 48109, USA

Self-reproduction in cellular automata is discussed with reference to the models of von Neumann and Codd. The conclusion is drawn that although the capacity for universal construction is a *sufficient* condition for self-reproduction, it is not a *necessary* condition. Slightly more "liberal" criteria for what constitutes genuine self-reproduction are introduced, and a simple self-reproducing structure is exhibited which satisfies these new criteria. This structure achieves its simplicity by storing its description in a dynamic "loop", rather than on a static "tape".

1. Introduction

Cellular automata were conceived by John von Neumann about 1950 as ideal structures for modeling self-reproducing "machines". The problem was stated by A.W. Burks [1]:

Von Neumann was interested in the general question: What kind of logical organization is *sufficient* for an automaton to be able to reproduce itself? The question is not precise and admits to trivial versions as well as interesting ones. Von Neumann had the familiar natural phenomenon of self-reproduction in mind when he posed it, but he was not trying to simulate the self-reproduction of a natural system at the levels of genetics and biochemistry. *He wished to abstract from the natural self-reproduction problem its logical form.* (emphasis added)

We will first review von Neumann's cellular self-reproducing automaton and a simplified model due to Codd. Then a further simplification will be presented: a small and compact "loop" embedded in a cellular automaton which is capable of self-reproduction.

Von Neumann's approach to the problem of self-reproduction was a classically logico-mathematical one: If self-reproduction *is* being carried out by a (highly complex) biochemical machine, then that machine's behavior is describable as a logical sequence of steps, i.e. as an algorithm. Now, if an algorithm can be performed by any machine at all, then there is a Turing machine which can perform the same algorithm. For this reason von Neumann set out to demonstrate the existence of a Turing machine which could effect its own reproduction. If such a Turing machine exists, it is entirely plausible that the processes by which living organisms reproduce themselves, and by implication, other processes on which life itself is based, are algorithmically describable and, therefore, that life itself is achievable by machines (a similar tenet is held by AI researchers with regard to the processes underlying intelligence).

Von Neumann was able to exhibit a universal Turing machine embedded in a cellular array using 29-states per cell and the 5-cell neighborhood. His Turing machine is suitably modified so that, as output, it can "construct" in the array any configuration which can be described on its input tape. Such a machine is called a *universal constructor*. His machine will construct any machine described on its input tape and, in addition, will also construct a copy of the input tape and attach it to the machine it has constructed. Now, self-reproduction follows as the special case where the machine described on the tape is the universal constructor itself. The result of the construction process is a copy of the universal constructor together with an input tape which contains its own description, which can then go on to construct a copy of itself, together with a copy of its own description, and so on indefinitely.

Note here that there are two levels of "automaton" in this construction: 1) the *cellular automaton* itself (the array); and 2) the *universal constructing automaton* which is embedded in the cellular automaton as a configuration of states. Thus a configuration can be an automaton itself. When referring to a "self-reproducing automaton" we will be referring to the embedded automaton, not to the cellular automaton.

Note also the manner in which the information on the input tape is used in two crucially different ways. First, the information on the tape is treated as *instructions to be interpreted* which, when executed, cause the construction of a machine somewhere else in the array. Second, the information on the tape is treated as *uninterpreted data*, which must be copied and attached to the new machine. These two different uses of information, interpreted and uninterpreted, are found in the process of natural self-reproduction as well, the former being the process of *translation*, and the latter *transcription*. It is interesting that the necessity for mechanisms which treat the same information in these two different ways resulted from von Neumann's research into the general problem of self-reproduction, independently of the discovery of the actual physical processes which the cell employs to carry them out.

The details of von Neumann's cellular construction were completed and published after his death by Arthur W. Burks [2], who worked with von Neumann on the logical design of one of the first stored program computers. As might be expected, von Neumann's self-reproducing "machine" is an enormously complex configuration, "built" in a cellular array, using the set of 29 states as logical building blocks, along the general lines of an early digital computer, replete with tape reading arms, "pulsers", clocks, encoders and decoders. Of course, the more complex the machine which is to accomplish the construction, the more complex the algorithm for building that machine will be, and, therefore, the longer the tape which contains the description of the machine. Thus, there is a genuine incentive for finding "simple" machines which are nonetheless still capable of self-reproduction.

For his Doctoral research at the University of Michigan, E.F. Codd [3] set out to reduce the complexity of von Neumann's machine. He was able to demonstrate a construction universal configuration which requires just 8-states per cell. Self-reproduction under Codd's construction is obtained as a special case of universal construction, just as it was in von Neumann's. In fact, the two machines behave in a very similar manner, the primary difference between them being that Codd's construction is obviously influenced by a careful consideration of the physiology of the nervous system in animals. Although simpler than von Neumann's, Codd's machine is still as complex as a modern digital computer, and, as far as is known, neither Codd's nor von Neumann's machines have actually been run under "real" simulation on a computer.

Thus, simpler machines than von Neumann's can be shown to be capable of reproducing themselves. The question then arises: How simple can a machine become while still retaining the capacity to reproduce itself? This question is the converse to von Neumann's question about *sufficient* organization, for it asks: What kind of logical organization is *necessary* for an automaton to be able to reproduce itself? This question is not precise either, for it also admits of trivial versions.

There are many examples of configurations being reproduced in cellular arrays which we would not want to call instances of self-reproduction. For example, if one takes the transition function to define modulo-two addition over the array, for an array of two-states per cell using the 5-cell neighborhood, one can observe simple configurations which appear to reproduce themselves. Starting with a single cell initialized to "on" and the rest to "off", a short time later there are five, isolated cells which are "on". Does this constitute self-reproduction? Clearly not. The initial cell "got reproduced" by the transition "physics", rather than by having reproduced itself. In general, there is no way in which a configuration consisting of just one cell could be said to reproduce itself. Any reproductive process going on here resides entirely in the transition rules, and not at all within the "configuration" itself.

In order to rule out such cases, as well as other trivially reproduced configurations, it has generally been required that any self-reproducing configuration must be capable of universal construction. This criterion, indeed, eliminates the trivial cases, but it also has the unfortunate consequence that it eliminates all naturally occurring self-reproducing systems as well, since none of these have been shown to be capable of universal construction. Furthermore, it is highly unlikely that the earliest self-replicating molecules, from which all living organisms are supposed to have been derived, were capable of universal construction, and we would not want to eliminate these from the class of truly self-reproducing configurations.

Thus, the criteria for what consitutes true self-reproduction need to be relaxed a bit, but no so far as to include the passive kind of reproduction mentioned above. It seems clear that we should take the "self" of "self-reproduction" seriously, and require of a configuration that the construction of the copy should be *actively directed* by the configuration itself. That is, responsibility for the production of the offspring should reside primarily within the sequences of actions undertaken by the parent structure. Note that we want to require that responsibility reside *primarily* with the parent structure itself, but not *totally*. This means that the structure *may* take advantage of certain properties of the transition function "physics" of the cellular space, as molecular reproduction most certainly does, but not to the extent that the structure is merely passively copied by mechanisms built into the transition function.

Von Neumann's work suggests an appropriate criterion, which is all the more appropriate because it is satisfied by molecular self-reproduction: the configuration must treat its stored information in the two different manners mentioned above: *interpreted*, as instructions to be executed (translation), and *uninterpreted*, as data to be copied (transcription).

In the remainder of this paper, we present an extremely simple configuration which can be embedded in a cellular automaton and which can effect its own reproduction by employing the processes of transcription and translation. Furthermore, self-reproduction is accomplished in a manner which does not depend on prior establishment of a capacity for universal construction.

2. A self-reproducing loop

Since the idea for this simple self-reproducing configuration came out of a study of the components of Codd's universal constructor, we need to go into some of the details of Codd's automaton. The basic structure upon which Codd bases his entire machine is the data-path (fig. 1). This consists of a string of cells in state 1 ("core" cells) surrounded by cells in state 2 ("sheath" cells). Data-paths, as their name implies, are capable of transmitting data in the form of "signals". A signal

```
2 2 2 2 2 2 2 2 2 2 2 2 2
1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2

          ( a )


2 2 2 2 2 2 2 2 2 2 2 2 2
1 1 1 1 1 0 7 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2

          ( b )


2 2 2 2 2 2 2 2 2 2 2 2 2
1 1 1 1 1 1 0 7 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2

          ( c )
```

Fig. 1. Signal propagation. (a) Codd's data-path: a string of "core" cells surrounded by "sheath" cells. (b) A "7 0" signal at time t. (c) The "7 0" signal at time $t + 1$. (Note: In all figures, cells not explicitly specified are in state 0.)

consists of a packet of two co-traveling states: the signal state itself (state 4, 5, 6 or 7) followed by the state 0. The packet travels with the signal state leading, and the 0 state trailing (figs. 1b, c). Data-paths may branch and fan out, in which case the signals are duplicated at the branching points, one copy proceeding down each branch (fig. 2). Signals are grouped into sequences, which are treated as instructions to effect certain actions. One such action, the extension of a data-path, is illustrated in fig. 3. The signal sequence "7 0 – 6 0", when it arrives at the cap at the end of a data-path, results in the data-path being extended by one cell. There is a similar signal sequence for retracting the data-path by one cell. There are also signal sequences which effect extension to the left or right instead of straight ahead, along with associated sequences to effect retraction of a left or right corner in a data-path.

```
          2 1 2
          2 1 2
          2 1 2
2 2 2 2 2 2 1 2 2 2 2 2 2
1 1 1 1 0 7 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2

        time  t


          2 1 2
          2 1 2
          2 1 2
2 2 2 2 2 2 1 2 2 2 2 2 2
1 1 1 1 1 0 7 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2

       time  t+1


          2 1 2
          2 1 2
          2 1 2
2 2 2 2 2 2 7 2 2 2 2 2 2
1 1 1 1 1 1 0 7 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2

       time  t+2


          2 1 2
          2 1 2
          2 7 2
2 2 2 2 2 2 0 2 2 2 2 2 2
1 1 1 1 1 1 1 0 7 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2

       time  t+3
```

Fig. 2. Signal duplication. What happens when a signal encounters a T-junction.

```
2 2 2 2 2 2 2 2 2 2 2 2 2
1 1 1 1 0 6 1 1 0 7 1 1 1 2
2 2 2 2 2 2 2 2 2 2 2 2 2

          ( a )


2 2 2 2 2 2 2 2 2 2 2 2 2
1 1 1 1 1 1 1 1 0 6 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2

          ( b )


2 2 2 2 2 2 2 2 2 2 2 2 2 2
1 1 1 1 1 1 1 1 1 1 1 1 1 1 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2

          ( c )
```

Fig. 3. Path extension. (a) A capped data-path with the "extend" signal sequence travelling along it. (b) The result of the "7 0" signal hitting the cap: the cap becomes a core cell. (c) The result of the "6 0" signal hitting the exposed core cell: the sheath is extended one cell and the path is capped.

The basic operation of Codd's machine is as follows. A data path is used as a tape reading arm, which is extended over a "tape", a linear string of states 0 and 1. The sequence of 1's and 0's on the tape is sensed by the tape arm and these sequences are decoded into a sequence of instructions. This sequence of instructions causes another data path, the "construction arm", to be extended into an empty area of the array and scanned back and forth over it, setting the cells it passes over to the proper states to form the configuration which is the machine described on the tape. Finally, a starting signal is injected into the new machine and the construction arm is withdrawn.

Codd uses a structure which he calls a "periodic emitter' as a basic timing element in his machine. The periodic emitter consists simply of a data path which folds back on itself to form a loop with a path leading away from it (fig. 4). Any signal traveling within the loop will be duplicated at the T-junction, with one copy going back around the loop, while the other copy heads off down the data-path leading away from the loop. Since the signal traveling around inside of the loop will arrive at the T-junction at regular intervals, a signal will be sent off down the data-path at regu-

```
  2 2 2 2 2 2
2 0 1 1 1 1 1 2
2 7 2 2 2 2 1 2
2 1 2     2 1 2
2 1 2     2 1 2
2 1 2 2 2 2 7 2 2 2 2 2 2 2 2 2 2 2 2 2
2 1 1 1 1 1 0 7 1 1 1 1 1 1 1 1 0 7 1 1
  2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Fig. 4. Periodic emitter. The two "7 0" signal sequences will keep cycling around the loop, sending a copy of the "7 0" signal off along the data-path every ten time steps.

```
  2 2 2 2 2 2
2 0 1 1 6 0 1 2
2 7 2 2 2 2 1 2
2 1 2     2 1 2
2 1 2     2 1 2
2 1 2 2 2 2 7 2 2 2 2 2 2 2 2 2 2 2 2 2
2 1 0 6 1 1 0 7 1 1 1 1 0 6 1 1 0 7 1 1 2
  2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Fig. 5. Path extension machine. As the two "extend" sequences cycle around the loop, they will form copies which will travel down the data-path, extending it indefinitely.

lar intervals as well, whence the name of this component.

Codd uses the periodic emitter solely for timing processes within his machine. However, this simple structure is an extremely important one, for it constitutes a *storage element*: signals cycling within the loop will remain cycling indefinitely, in a manner reminiscent of the serial delay line memories of the earliest stored program computers. Thus, one can use a loop of this sort to store a program dynamically, rather than in static form, as on a tape. This is of tremendous aid in the design of a self-reproducing machine because it eliminates the complex machinery associated with moving a read head back and forth over a static tape, and the simpler the machine, the simpler it is to reproduce. Furthermore, rather than having to decode the information on a tape to obtain the signal sequences necessary to effect the construction of a machine, the "program" which we store in the loop can simply *be* the proper signal sequence to effect the construction, the same sequence which would have been generated as a result of decoding a sequence of 1 and 0 marks on a static tape. Thus, we can also eliminate the complex decoding and signal generation circuitry from our machine.

Thus, from these cyclic storage loops, we can build *many* machines, simple or complex, which can perform any task that we desire, simply by storing the appropriate sequence of instructions in a loop of the appropriate size. For example, if we put the path extension sequence into a loop, we get a machine which will extend its data-path indefinitely (fig. 5).

Can we make such a loop reproduce itself? It is not immediately obvious that we can, because it may not be possible to fit the instructions to build a loop *into* a loop of that size. Indeed, it does not seem possible to do this with Codd's signal sequences, since he requires the sequence "7 0 1 1 6 0" just to extend the path by one cell. If it takes *six* cells to store the information to construct a *single* cell of the new machine, how could a loop store enough information to build a structure identical in size to itself? The answer is that if we make the storage loop a perfect square, we need only store the instructions necessary to build one side and one corner of the loop. As these instructions cycle around the loop four times, they will repeat the process of building a side and a corner four times. Thus we can reduce the space requirement for the storage of the instructions by a factor of four.

As it turns out, using Codd's signal sequences, there is still not enough room in a loop to store the instructions to build one side and one corner, as they are too long. However, since we are not interested in maintaining the capacity of universal construction, we alter the meaning of some of his signals, making them individually more powerful, at the expense of making them collectively less general. Altering the meaning of signals is accomplished, not by changing *configurations* in the array, as we have been describing above, but by changing the *transition rules* which control the *behavior* of the configurations in the array. By this technique, we shorten the overall sequence necessary to build one side and one corner sufficiently so that it can now fit into a loop of the size which it constructs. Thus, where Codd requires the sequence "7 0 – 6 0" to extend the data-path by one cell, we redefine the "7 0" signal to cause this extension by itself. Furthermore, where signals in Codd's machine had to be spaced four cells apart, we allow signals to be only three cells apart. We also allow a left path extension to be accomplished by two successive "4 0" signals, where Codd required the sequence "4 0–4 0–5 0–6 0". We also allow a path which is extended into the side of another path to "fuse" with that path, forming a new T-junction. After all of these changes have been made, we have freed several of Codd's signals from any vestiges of their former meanings and can therefore apply them to other ends. The altered set of transition rules is listed in table I.

The loop configuration which emerges after all of this modification is shown in fig. 6. The instruction sequence for self-reproduction is:

```
  2 2 2 2 2 2 2 2
2 1 7 0 1 4 0 1 4 2
2 0 2 2 2 2 2 2 0 2
2 7 2         2 1 2
2 1 2         2 1 2
2 0 2         2 1 2
2 7 2         2 1 2
2 1 2 2 2 2 2 2 1 2 2 2 2 2
2 0 7 1 0 7 1 0 7 1 1 1 1 1 2
  2 2 2 2 2 2 2 2 2 2 2 2 2

        TIME = 0
```

Fig. 6. Self-reproducing loop.

7 0–7 0–7 0–7 0–7 0–7 0–4 0–4 0.

The six "7 0" signals will extend the construction arm by six cells, while the two "4 0" signals will build a left hand corner at the end of the arm. After each cycle of the instructions around the loop,

Table I
Transition function table for self-reproducing loops

CTRBL->I	CTRBL->I	CTRBL->I	CTRBL->I	CTRBL->I
00000->0	02527->1	11322->1	20242->2	30102->1
00001->2	10001->1	12224->4	20245->2	30122->0
00002->0	10006->1	12227->7	20252->0	30251->1
00003->0	10007->7	12243->4	20255->2	40112->0
00005->0	10011->1	12254->7	20262->2	40122->0
00006->3	10012->1	12324->4	20272->2	40125->0
00007->1	10021->1	12327->7	20312->2	40212->0
00011->2	10024->4	12425->5	20321->6	40222->1
00012->2	10027->7	12426->7	20322->6	40232->6
00013->2	10051->1	12527->5	20342->2	40252->0
00021->2	10101->1	20001->2	20422->2	40322->1
00022->0	10111->1	20002->2	20512->2	50002->2
00023->0	10124->4	20004->2	20521->2	50021->5
00026->2	10127->7	20007->1	20522->2	50022->5
00027->2	10202->6	20012->2	20552->1	50023->2
00032->0	10212->1	20015->2	20572->5	50027->2
00052->5	10221->1	20021->2	20622->2	50052->0
00062->2	10224->4	20022->2	20672->2	50202->2
00072->2	10226->3	20023->2	20712->2	50212->2
00102->2	10227->7	20024->2	20722->2	50215->2
00112->0	10232->7	20025->0	20742->2	50222->0
00202->0	10242->4	20026->2	20772->2	50224->4
00203->0	10262->6	20027->2	21122->2	50272->2
00205->0	10264->4	20032->6	21126->1	51212->2
00212->5	10267->7	20042->3	21222->2	51222->0
00222->0	10271->0	20051->7	21224->2	51242->2
00232->2	10272->7	20052->2	21226->2	51272->2
00522->2	10542->7	20057->5	21227->2	60001->1
01232->1	11112->1	20072->2	21422->2	60002->1
01242->1	11122->1	20102->2	21522->2	60212->0
01252->5	11124->4	20112->2	21622->2	61212->5
01262->1	11125->1	20122->2	21722->2	61213->1
01272->1	11126->1	20142->2	22227->2	61222->5
01275->1	11127->7	20172->2	22244->2	70007->7
01422->1	11152->2	20202->2	22246->2	70112->0
01432->1	11212->1	20203->2	22276->2	70122->0
01442->1	11222->1	20205->2	22277->2	70125->0
01472->1	11224->4	20207->3	30001->3	70212->0
01625->1	11225->1	20212->2	30002->2	70222->1
01722->1	11227->7	20215->2	30004->1	70225->1
01725->5	11232->1	20221->2	30007->6	70232->1
01752->1	11242->4	20222->2	30012->3	70252->5
01762->1	11262->1	20227->2	30042->1	70272->0
01772->1	11272->7	20232->1	30062->2	

```
Neighborhoods are read as follows (rotations are not listed):

                T
              L C R        ==>        I
                B
```

another side and corner has been built (fig. 7). This process continues until the arm has closed back on, and "fused", with itself. Fig. 8 shows the way in which the collision of signals at the newly formed junction results in the generation of two new signals: a "5" signal, traveling back toward the parent loop, pulling the "umbilical cord" behind it back into the body of the parent loop; and a "6 0" signal traveling on into the offspring loop. This "6 0" signal does *not* turn the corner in the offspring loop when it hits it, but instead breaks through the sheath and initiates the construction of a new construction arm for the offspring loop. Meanwhile, a "7 0" signal was sent on around the corner, in place of the "6 0" signal which broke through the sheath, patching up the sequence of instructions, which is now left cycling around the offspring loop. Meanwhile, back in the parent loop, the "5" signal progresses on to the next corner, via the sheath, where it will initiate the construction of a new construction arm for the parent loop. The result of all of this activity is that, after 151 time steps, we are left with an offspring loop which is an exact copy of its parent at time 0 (fig. 9). Thus the loop has succeeded in reproducing itself.

Since the offspring loop is an exact duplicate of the parent loop, it will behave in exactly the same manner as its parent and construct an offspring loop to its "right". Notice that the parent loop is about to construct a second offspring loop "above" itself. Although not explicitly required by our criteria, one feels intuitively that to effect true reproduction the result should be *two* copies which are identical both physically *and* behaviorally to the original. Thus we want to leave the parent with the capacity to reproduce itself after it has produced an offspring which can do so.

As can be seen, after each offspring has been produced, the construction arm is moved 90 degrees counterclockwise, and a new offspring will be constructed in this area of the array. However, this process cannot continue indefinitely without a loop eventually trying to build an offspring in an area already occupied by another loop. When a loop encounters another loop residing in a potential offspring site, the loop retracts its arm back into the corner, actually blocking its own core-path with a sheath-cell. When the cycling instructions run into this sheath-cell blockade, they are simply erased, one by one, until the loop is left empty of instructions.

Thus, over a period of time, there will emerge an expanding *colony* of loops growing out into the array. Figure 10 shows seven generations of the growth of the colony. The colony consists of a

```
  2 2 2 2 2 2 2 2
2 4 0 1 1 1 1 1 7 2
2 1 2 2 2 2 2 2 0 2
2 0 2         2 1 2
2 4 2         2 7 2
2 1 2         2 0 2
2 0 2         2 1 2                   2
2 7 2 2 2 2 2 2 7 2 2 2 2 2 2 2 2 2 2 1 2
2 1 0 7 1 0 7 1 0 7 1 0 7 1 0 7 1 1 1 1 2
  2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

              TIME = 35
```

```
  2 2 2 2 2 2 2 2                   2 2
2 7 0 1 7 0 1 7 0 2               2 1 1 2
2 1 2 2 2 2 2 2 1 2                 2 1 2
2 1 2         2 7 2                 2 1 2
2 1 2         2 0 2                 2 1 2
2 1 2         2 1 2                 2 7 2
2 1 2         2 7 2                 2 0 2
2 0 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 1 2
2 4 1 0 4 1 0 7 1 0 7 1 0 7 1 0 7 1 0 7 2
  2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

              TIME = 70
```

```
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2
2 0 1 7 0 1 7 0 1 2   2 1 1 1 1 7 0 1 7 2
2 7 2 2 2 2 2 2 7 2   2 1 2 2 2 2 2 2 0 2
2 1 2         2 0 2     2           2 1 2
2 0 2         2 1 2                 2 7 2
2 7 2         2 4 2                 2 0 2
2 1 2         2 0 2                 2 1 2
2 0 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 7 2
2 7 1 1 1 1 1 0 4 1 0 4 1 0 7 1 0 7 1 0 2
  2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

              TIME = 105
```

```
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2
3 0 1 1 1 1 1 7 0 2   2 1 7 0 1 7 0 1 4 2
2 4 2 2 2 2 2 2 1 2   2 0 2 2 2 2 2 2 0 2
2 1 2         2 7 2   2 7 2         2 1 2
2 0 2         2 0 2   2 1 2         2 4 2
2 4 2         2 1 2   2 1 2         2 0 2
2 1 2         2 7 2     2           2 1 2
2 0 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 1 2
2 7 1 0 7 1 0 7 1 0 7 1 0 7 1 0 7 1 1 1 2
  2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

              TIME = 120
```

Fig. 7. Construction of offspring loop. With every cycle of the instructions around the parent loop, a side and a corner of the offspring loop are built.

```
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 3
2 1 1 7 0 1 7 0 1 2   2 7 0 1 4 0 1 4 0 2
2 1 2 2 2 2 2 2 7 2   2 1 2 2 2 2 2 2 1 2
2 1 2         2 0 2   2 0 2         2 1 2
2 1 2         2 1 2   2 7 2         2 1 2
2 0 2         2 7 2   2 1 2         2 1 2
2 4 2         2 0 2   2 1 2         2 1 2
2 1 2 2 2 2 2 2 1 2 2 2 1 2 2 2 2 2 2 7 2
2 0 4 1 0 7 1 0 7 1 0 7 1 0 7 1 0 7 1 0 2
  3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

                TIME = 124
```

```
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2
2 1 7 0 1 7 0 1 7 2   2 0 1 4 0 1 4 0 1 2
2 1 2 2 2 2 2 2 0 2   2 7 2 2 2 2 2 2 1 2
2 1 2         2 1 2   2 1 2         2 1 2
2 1 2         2 7 2   2 0 2         2 1 2
2 1 2         2 0 2   2 7 2         2 1 2
2 0 2         2 1 2   2 1 2         2 7 2
2 4 2 2 2 2 2 2 7 2 2 2 1 2 2 2 2 2 2 0 2
2 1 0 4 1 0 7 1 0 7 1 0 0 1 0 7 1 0 7 1 2
  2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

                TIME = 125
```

```
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2
2 7 0 1 7 0 1 7 0 2   2 1 4 0 1 4 0 1 1 2
2 1 2 2 2 2 2 2 1 2   2 0 2 2 2 2 2 2 1 2
2 1 2         2 7 2   2 7 2         2 1 2
2 1 2         2 0 2   2 1 2         2 1 2
2 1 2         2 1 2   2 0 2         2 7 2
2 1 2         2 7 2   2 7 2         2 0 2
2 0 2 2 2 2 2 2 0 2 2 2 1 2 2 2 2 2 2 1 2
2 4 1 0 4 1 0 7 1 0 7 5 0 6 1 0 7 1 0 7 2
  2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

                TIME = 126
```

```
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2
2 0 1 7 0 1 7 0 1 2   2 4 0 1 4 0 1 1 1 2
2 7 2 2 2 2 2 2 7 2   2 1 2 2 2 2 2 2 1 2
2 1 2         2 0 2   2 0 2         2 1 2
2 1 2         2 1 2   2 7 2         2 7 2
2 1 2         2 7 2   2 1 2         2 0 2
2 1 2         2 0 2   2 0 2         2 1 2
2 1 2 2 2 2 2 2 1 2 2 2 7 2 2 2 2 2 2 7 2
2 0 4 1 0 4 1 0 7 1 5 2 1 0 6 1 0 7 1 0 2
  3 2 2 2 2 2 2 2 2 2   2 2 2 2 2 2 2 2

                TIME = 127
```

```
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2
2 1 7 0 1 7 0 1 7 2   3 0 1 4 0 1 1 1 1 2
2 0 2 2 2 2 2 2 0 2   2 4 2 2 2 2 2 2 1 2
2 7 2         2 1 2   2 1 2         2 7 2
2 1 2         2 7 2   2 0 2         2 0 2
2 1 2         2 0 2   2 7 2         2 1 2
2 1 2         2 1 2   2 1 2         2 7 2
2 1 2 2 2 2 2 2 7 2   2 0 2 2 2 2 2 2 0 2
2 1 0 4 1 0 4 1 0 5   2 7 1 0 6 1 0 7 1 2
  2 2 2 2 2 2 2 2 2     2 2 2 2 2 2 2 2

                TIME = 128
```

```
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2
2 7 0 1 7 0 1 7 0 2   2 1 4 0 1 1 1 1 1 2
2 1 2 2 2 2 2 2 1 2   2 0 2 2 2 2 2 2 7 2
2 0 2         2 7 2   2 4 2         2 0 2
2 7 2         2 0 2   2 1 2         2 1 2
2 1 2         2 1 2   2 0 2         2 7 2
2 1 2         2 7 2   2 7 2         2 0 2
2 1 2 2 2 2 2 2 0 5   2 1 2 2 2 2 2 2 1 2
2 1 1 0 4 1 0 4 1 2   2 0 7 1 0 6 1 0 7 2
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2

                TIME = 129
```

```
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2
2 0 1 7 0 1 7 0 1 2   2 4 0 1 1 1 1 1 7 2
2 7 2 2 2 2 2 2 7 2   2 1 2 2 2 2 2 2 0 2
2 1 2         2 0 2   2 0 2         2 1 2
2 0 2         2 1 2   2 4 2         2 7 2
2 7 2         2 7 2   2 1 2         2 0 2
2 1 2         2 0 5   2 0 2         2 1 2
2 1 2 2 2 2 2 2 1 2   2 7 2 2 2 2 2 2 7 2
2 1 1 1 0 4 1 0 4 2   2 1 0 7 1 0 6 1 0 2
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2

                TIME = 130
```

```
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2
2 1 7 0 1 7 0 1 7 2   3 0 1 1 1 1 1 7 0 2
2 0 2 2 2 2 2 2 0 2   2 4 2 2 2 2 2 2 1 2
2 7 2         2 1 2   2 1 2         2 7 2
2 1 2         2 7 2   2 0 2         2 0 2
2 0 2         2 0 5   2 4 2         2 1 2
2 7 2         2 1 2   2 1 2         2 7 2
2 1 2 2 2 2 2 2 4 2   2 0 2 2 2 2 2 2 0 2
2 1 1 1 1 0 4 1 0 3   2 7 1 0 7 1 0 6 1 2
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2

                TIME = 131
```

```
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2
2 7 0 1 7 0 1 7 0 2   2 1 1 1 1 1 7 0 1 2
2 1 2 2 2 2 2 2 1 2   2 0 2 2 2 2 2 2 7 2
2 0 2         2 7 2   2 4 2         2 0 2
2 7 2         2 0 5   2 1 2         2 1 2
2 1 2         2 1 2   2 0 2         2 7 2
2 0 2         2 4 2   2 4 2         2 0 2
2 7 2 2 2 2 2 2 0 2   2 1 2 2 2 2 2 2 1 2
2 1 1 1 1 1 0 4 1 2   2 0 7 1 0 7 1 0 3 2
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2

                TIME = 132
```

```
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2
2 0 1 7 0 1 7 0 1 2   2 1 1 1 1 7 0 1 7 2
2 7 2 2 2 2 2 2 7 2   2 1 2 2 2 2 2 2 0 2
2 1 2         2 0 5   2 0 2         2 1 2
2 0 2         2 1 2   2 4 2         2 7 2
2 7 2         2 4 2   2 1 2         2 0 2
2 1 2         2 0 2   2 0 2         2 1 2
2 0 2 2 2 2 2 2 1 2   2 4 2 2 2 2 2 2 7 2
2 7 1 1 1 1 1 0 4 2   2 1 0 7 1 0 7 1 0 6
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2

                TIME = 133
```

```
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2
2 1 7 0 1 7 0 1 7 2   2 1 1 1 7 0 1 7 0 2
2 0 2 2 2 2 2 2 0 5   2 1 2 2 2 2 2 2 1 2
2 7 2         2 1 2   2 1 2         2 7 2
2 1 2         2 4 2   2 0 2         2 0 2
2 0 2         2 0 2   2 4 2         2 1 2
2 7 2         2 1 2   2 1 2         2 7 2
2 1 2 2 2 2 2 2 4 2   2 0 2 2 2 2 2 2 0 2
2 0 7 1 1 1 1 1 0 3   2 4 1 0 7 1 0 7 1 1 3
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2 2

                TIME = 134
```

Fig. 8. Separation of offspring. Collision of signals at new junction generates signals which separate the loops and initiate construction of new construction arms in both parent and offspring loops.

```
                2
              2 1 2
              2 7 2
              2 0 2
              2 1 2
  2 2 2 2 2 2 2 7 2     2 2 2 2 2 2 2 2
2 1 1 1 7 0 1 7 0 2   2 1 7 0 1 4 0 1 4 2
2 1 2 2 2 2 2 2 1 2   2 0 2 2 2 2 2 2 0 2
2 1 2         2 7 2   2 7 2         2 1 2
2 0 2         2 0 2   2 1 2         2 1 2
2 4 2         2 1 2   2 0 2         2 1 2
2 1 2         2 7 2   2 7 2         2 1 2
2 0 2 2 2 2 2 2 0 2   2 1 2 2 2 2 2 2 1 2 2 2 2 2
2 4 1 0 7 1 0 7 1 2   2 0 7 1 0 7 1 0 7 1 1 1 1 1 2
  2 2 2 2 2 2 2 2       2 2 2 2 2 2 2 2 2 2 2 2 2

              TIME = 151
```

Fig. 9. Self-reproduction completed. After 151 time steps, the loop has reproduced itself exactly.

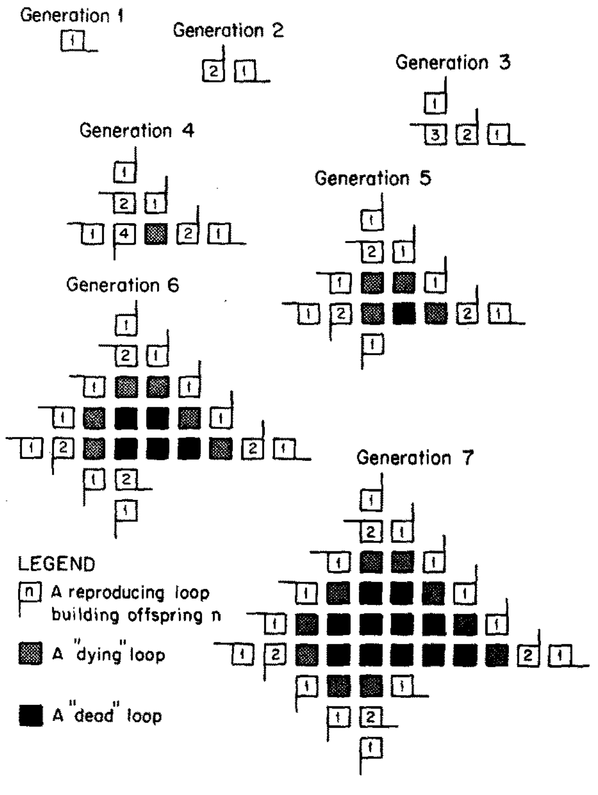

Fig. 10. Growth of loop colony. Seven generations of growth in a colony of loops.

reproductive fringe surrounding a growing core of empty loops, much like the growth of a coral reef. In an infinite cellular array, the colony would keep growing indefinitely.

It is of interest to note that, although each loop is identical in structure, the loops will behave differently in different environments. Specifically, the original loop (the "Adam" loop) will reproduce four times, whereas each of its immediate offspring will reproduce only twice. The first offspring of an offspring of the "Adam" loop will reproduce twice, while the second offspring will reproduce only once. Thus identical loops will behave differently in different environments, the determining environmental factor being the concentration of loops in the immediate neighborhood of a loop. Of course, any loop could become an "Adam" loop if the other loops in its neighborhood were suddenly cleared away.

It is also of interest to note the various stages in the "life cycle" of a loop. Each loop has a "fetal" stage: the period during which the arm of the parent is being extended and bent around to form the offspring, the arm eventually becoming an "umbilical cord" which serves as a channel for getting the cycling instructions to the growing end of the "looplet". Following this fetal stage is the "adolescent" stage, which lasts from "birth", the moment when the umbilical cord is disconnected, until the offspring's construction arm has been completed. Then follows the period of reproductive "adulthood", during which the loop produces as many offspring as the environment will allow. As soon as the loop encounters another loop occupying a potential offspring site, the loop enters into its decline as its cycling instructions are slowly erased. The final result is an empty or "dead" loop.

A significant difference between these loops and von Neumann's or Codd's machines is that the latter involve a *passive* description on the tape being acted upon by a *dynamic* interpreter, whereas the loops involve a *dynamic* description acting upon an essentially *passive* interpreter (the body of the loop itself). Despite this reversal of roles on the part of the description and the interpreter, the two operations of translation and transcription remain well defined. *Translation* is accomplished when the instruction signals are "executed" as they reach the end of the construction arms, and upon the collision of signals with other signals. *Transcription* is accomplished by the duplication of signals at the T-junctions.

These loops are also significantly simpler than the machines of von Neumann or Codd. If we imagine a continuous scale of complexity of self-reproducing entities, with one end representing the simple, marginally self-facilitating copying processes which must have been supported by the physics and chemistry (the "transition rules") of the early pre-biological "soup", and the other end representing the highly complex mechanisms of molecular self-reproduction, as well as the universal constructors of von Neumann and Codd, these loops occupy a spot somewhere in the middle ground. They are sufficiently complex so as to be quite clearly self-reproductive, yet, at the same time, they are sufficiently simple so as to constitute "believable" extensions of simpler copying processes.

We return now to consider our criteria for self-reproduction and ask if the loops satisfy all of our requirements. In the first place, the responsibility for the production of the offspring quite obviously resides with the sequences of actions taken by the parent loop. The cycling instruction sequence *actively directs* every phase of the construction of the offspring. In the second place, the reproduction is effected by using the information contained in the cycling description in two different ways: as instructions to be translated, and as data to be transcribed. Thus, the loops satisfy all of the criteria and must be considered as truly self-reproducing structures.

3. Conclusion

We have now demonstrated the existence of a simple self-reproducing structure. We conclude with a brief summary of the novel features of this structure.

In the first place, the loops are very simple structures. Whereas von Neumann's and Codd's machines occupy many tens of thousands of cells, a loop will fit in a rectangular area of just 10 by 15 cells.

In the second place, the loop's reproduction does not depend on any demonstrated capacity for universal construction. It was argued that, although universality is a *sufficient* condition for self-reproduction, it is not a *necessary* condition. This is a primary factor in the simplicity of the loops.

Finally, it was shown that the loops actively direct their reproduction, employing both transcription and translation in the process, and thus reproduce non-trivially. This was shown to be the case despite the reversal of dynamic and static roles by the description and the interpreter.

Acknowledgements

This work was supported in part by the National Science Foundation, Grant No. MCS78-26016. I would like to thank Arthur W. Burks and John Holland of the University of Michigan, and Tom Toffoli of MIT, for their many helpful suggestions and comments.

References

[1] A.W. Burks, ed., Essays on Cellular Automata (Univ. of Illinois Press, Illinois, 1968) p. xv.

[2] J. von Neumann, The Theory of Self-Reproducing Automata, A.W. Burks, ed. (Univ. of Illinois Press, Illinois, 1966).

[3] E.F. Codd, Cellular Automata (Academic Press, New York, 1968).