Alvy Ray Smith
1001 West Cutting Blvd., Point Richmond, CA 94804

Simple Nontrivial Self-Reproducing Machines

A simple and brief proof of the existence of *nontrivial* self-reproducing machines, as cellular automata (CA) configurations, is presented, which relies only on computation universality. Earlier proofs are book length and rely on "construction universality." Furthermore, simple CA are shown to support nontrivial self-reproduction—hence, simultaneously simple and nontrivial. Nontriviality is guaranteed by the requirement that the machine which reproduces itself is also a universal computer. Biological relevance—or non-relevance—is also briefly discussed, as is trivial self-reproduction, called self-replication.

INTRODUCTION

The purpose of this note is to increase awareness of a cellular automata (CA)[1] result which is over 20 years old, but previously unavailable except in my Stanford

[1]There has been a revival in interest in CA theory since the combined advent of chaos, fractals, and interactive computer graphics in the 1980s, inspired principally by Wolfram and the other authors.[39] There is a surprising ignorance by much of the post-revival literature of the hundreds of papers written during the preceding 20 years or so. Part of the problem is the lack of knowledge that CA had many other names then: iterative arrays, tessellation automata, cellular spaces, modular

Ph.D. dissertation,[25] abbreviated in an old conference proceedings,[24] and hinted at in exercises in Arbib.[5] This is a *brief* proof of the existence of self-reproducing machines which are nontrivial in the sense that they are capable of computing any computable function. In fact, several different proofs are given.

This is to be compared to numerous lengthy or complex proofs. John von Neumann introduced the concept in a book[35] which also created the field of CA theory (from a suggestion by Stanislaw Ulam). We will use the term "machine" in the same sense as von Neumann—a finite collection of non-quiescent cells in a CA. Both his CA (29-state cells) and his self-reproducing configuration (40,000 or so nonquiescent cells) were complex and inspired others[3,6,10,33] to simplify. Von Neumann's lengthy proof—book length—is constructive. The other proofs just mentioned are also constructive and book length, except for Arbib[3] which is short but uses very complex cells. The proof exhibited here is an existence proof only, but it is just two pages long. The constructions of the constructive proofs are very tedious. This plus the fact that the principal result in all cases is that self-reproduction is logically possible makes an existence proof sufficient. And, as we shall see, the existence proof herein is actually stronger than that term might suggest, it being theoretically possible to compute a construction from the proof.

More importantly the proof here reduces the problem of self-construction to a computation problem, which means that no machinery beyond ordinary computation theory is required for self-reproduction. This is the real reason for the brevity of the proof. It will be seen to turn on invocation of the famous Recursion Theorem of recursive function theory (see Rogers,[23] for example).

The result and its relation with the literature is brought up-to-date.

RECURSIVE FUNCTION THEORY

The branch of mathematics which deals with those computations that digital computers can perform is called recursive function theory. This theory tells us that there exist "effective enumerations" of the partial recursive functions; this simply means that all computer programs can be listed by some one computer program (which itself must be in the list, of course). Let's call this list "the List." Then the ith program (partial recursive function) in the List is denoted List(i).[2]

Two well-known theorems used herein are these:

arrays, polyautomata, etc. The references herein can be used as pointers into this literature (e.g., see Smith[30]).

[2] As usual a *total* function is one defined for every element of its (input) domain, whereas a *partial* function may be undefined for some elements of its domain. A total recursive function corresponds to a computer program which halts on every input, and a partial recursive function to one that may not halt on some inputs (for example, it might go into an infinite loop).

THE UNIVERSAL TURING MACHINE THEOREM In the List of all programs is a special program, $U = \text{List}(j)$, and a total recursive function e of two variables such that, for an arbitrary program $P = \text{List}(i)$ and an arbitrary input x to that program, $U(\, e(P,\, x)\,) \;=\; P(x)$ if $P(x)$ is defined, and is undefined if $P(x)$ is undefined. We will call U in this case a *universal Turing machine function* U and e the *encoder function* associated with U. In other words, one of the programs in the List of programs can simulate all others, given an encoding of each program and its input. U is the mathematical representation of today's digital computer. It was Alan Turing who made this wonderful discovery (in 1937) of a universal computer and initiated the digital computer revolution, so we name the function after him.

We assume that an encoder function is simple—that it is not complex enough to hide a universal computer, for example. We also assume that an encoder function is total recursive on (program, input) pairs. In fact, we assume it is *birecursive*, meaning that it can be uniquely decoded in a computable way.

In practice, a generalization of the above is allowed. Thus, an encoding of an arbitrary program and its input is given to a universal computer. The universal computer simulates the given program on its given input and generates the output of the simulated program. In the Universal Turing Machine Theorem statement above, the output of the universal computer is exactly the output of the simulated program. In practice, a machine is still considered universal if it outputs an encoding of the simulated output rather than the simulated output itself. As for the program/input encoding function, the output encoding function is assumed to be simple. We shall call universal computers *decodeless* if they do not require an output encoding function.

It is worth pointing out here that any universal computer which is not decodeless can be made so by appending a "termination subroutine" to its program which decodes the encoded output before halting. This is possible because we assume decoding is computable. Clearly, this larger program is still a program and, hence, in the List. So there are lots of universal computer programs (or functions) in the List. The next important theorem says this more formally.

THE RECURSION THEOREM If h is a total recursive function from programs into programs, then there is a fixed-point program P such that $P \;=\; h(P)$. This is just a way of saying that any computation shows up in the List many times—that the List is redundant, certainly in any computable way (represented by h)—or that there are many different programs that ultimately compute the same thing.

It's not hard to understand why. Consider a program which performs the computation of adding 1 to its input. There is another program which does the same thing but (stupidly) executes a subroutine which counts to a million before exiting. Since any number can be substituted for "million" in the preceding sentence and each resulting program is different, then the List must contain them all or a countable infinity of programs for adding 1. In this case, h is the function that appends to any program the subroutine which does nothing but count to a million. The h introduced in this paper is far more interesting and the resulting fixed-point program of the Recursion Theorem far more profound—it is crucial to our desired result.

Since P in the Recursion Theorem above can be computed (e.g., see Arbib[5]), our existence proof will be constructive, in this theoretical sense.

Alan Turing invented, in the course of his studies leading to the universal computer theorem, a theoretical computer now called a *Turing machine*. This very simple device is instrumental in our results here, and details will be provided in a later section. It is mentioned here because each program in the List can be thought of as a Turing machine. In other words, a list of all Turing machines could be the List we refer to above, and both of the important theorems above apply to it. In fact, this is why the first theorem is called the Universal Turing Machine Theorem. The universal Turing machine is that Turing machine in the List which simulates all other Turing machines in the List. Several leading mathematicians in the 1930s and 1940s spent considerable effort proving that several apparently different ways of describing computation were all equivalent. We therefore feel free today to use "program," "Turing machine," "effective computation," and "partial recursive function" interchangeably for the same concept although the formalisms are often different.

CONSTRUCTION VS. COMPUTATION

Usually it is only the computational abilities of CA which are investigated. It is also of interest to study the "constructional" abilities—the construction of one configuration by another. This was one of the original motivations for the study of CA. As mentioned above, von Neumann[35] introduced the CA, conceived as an infinite chessboard, each square, or "cell," of which represents a copy of a single finite automaton (computer), as an environment in which to study the logical intricacies of biological reproduction. This paper is also along these lines.

It is not at all clear what "construction" should be defined to be. In fact, Holland[12] has given arguments which show that Moore-type CA are inherently incapable of supporting construction in the full generality that term might support—namely, construction of, or simulation of, arbitrary networks of finite automata.[3] This is no surprise in view of the existence of Garden-of-Eden configurations (patterns which cannot arise from computation, but can only exist at time zero by external programming of the CA) in even trivial CA.[25] Lieblein[15] has characterized the sequences of configurations which can be attained in Moore-type CA. We shall be interested in sequences which are readily realized in a Moore-type CA and will not even find it necessary to define construction—computation will be sufficient.

The type of construction process which is treated in this paper is called *self-reproduction*[20] where the definition of this term is assumed: Let c_0 be an initial

[3]A *Moore-type*, or sequential, CA is one with memory at each cell so that there is a delay between any input and its associated output. This is distinguished from *Mealy-type* CA which allow instantaneous (combinational) output from inputs at each cell. The von Neumann CA and all others considered in this paper are Moore-type.

configuration or subconfiguration in some (infinite) CA. Then c_0 self-reproduces in the (Moore) sense if, at some time $t > 0$, there exist at least two disjoint copies of c_0; at time $t' > t$, there exist at least three disjoint copies of c_0; etc. To say there exists two disjoint copies of a configuration c in configuration c' means, of course, that there is a translation δ such that $c' = c \cup \delta c$. Two configurations are *disjoint*, of course, if their non-quiescent cells are disjoint.

As Moore points out, this definition permits trivial CA phenomena to be interpreted as self-reproduction. For example, the configuration consisting of a single non-quiescent cell (a single 1 on a background of 0's) self-reproduces in the one-dimensional CA with K_1 template (a cell and one of its nearest neighbors—see Smith[27] for details) and transition function f which maps the current state of a cell, given the current state of its neighbor, to next-state 1 (that is, $f(q_0, q_1) = 1$ except for $f(0, 0)$ which by definition of quiescent background must be 0). Thus, care will be exercised to ensure the nontriviality of any self-reproducing configuration introduced in this paper. In particular, we will insist, as is the historical precedent, that a self-reproducing configuration be a universal computer. This definition of nontriviality is perhaps overly restrictive for biological purposes, but we as yet do not have a more lifelike complexity requirement.

TURING MACHINE SIMULATION

The environment of the self-reproducing machines to be introduced below will be CA of the simple Turing-machine simulation type introduced in Smith.[24,26] We first review Turing machines and then outline the simple simulations of them by CA we require.

A Turing machine is a conceptually simple device consisting of a one-dimensional *tape* divided lengthwise into squares and a *head* which scans along the tape in either direction. The tape is assumed infinite in extent and each square is initially blank, but for a finite input section. The head has a finite amount of memory, expressed as a finite number of states the memory can be in. The head can be thought of as containing the program to be executed by the Turing machine. At any one step, the Turing machine head can read or write the square currently being scanned and can then move either one square left or right. It can read or write a finite set of symbols, of which one is the blank, on its tape. We will refer to an (m, n) Turing machine M and mean an m-symbol, n-state Turing machine. It is surprising that this simple device can compute any computable function but this is what the Universal Turing Machine Theorem says. Minsky has found the simplest known universal Turing machine to be a (4, 7) Turing machine; he has also found a (6, 6) universal Turing machine.[19] These are not decodeless universal computers; they assume the simulated output tape is encoded.

The notion of simulating a Turing machine by a CA is straightforward (and rediscovered numerous times): A row of CA cells simulates the squares on a Turing machine tape while at the same time simulating the head of the Turing machine. The state set of the CA cell is chosen to have two coordinates: one simulates the symbol on the Turing machine tape; the other simulates the state of the Turing machine head or the absence of the head at that cell. Appropriate choice of transition function operating on the nearest-neighbor neighborhood makes it easy to directly simulate the Turing machine. It is easy to show that an (m, n) Turing machine can be simulated by a one-dimensional CA with $m \times (n+1)$ states per cell. Without working too much harder, this can be improved to $m + 2n$ states per cell.[26] Thus, the (4, 7) universal computer (or the (6, 6)) can be simulated by an 18-state, 3-neighbor CA.[4] We will say that Turing machine M is *wired into* CA Z_M or that Z_M has M *wired in*.

For notational convenience, the following devices will be employed:

$$x \rightarrow_P y \tag{1}$$

means that Turing machine program P acting on initial tape x halts with y (and nothing else but blank squares) on the tape as its final result.

$$x \rightarrow_P y \rightarrow_P y' \rightarrow_P \ldots \tag{2}$$

means that Turing machine program P acting on initial tape x alters the tape until y is on the tape at some time $t > 0$. P proceeds to compute on this tape until y' appears on it at some time $t' > t$; and so on. This is a way of representing the temporal sequence of tapes generated by a Turing machine treated as generator of recursively enumerable sets instead of as a computer of partial recursive functions.

$$\uparrow x \tag{3}$$

indicates that a Turing machine control head is scanning the leftmost (nonblank) symbol in the string x written on its tape.

Exploiting the ease of simulation of Turing machines by CA leads directly to self-reproducing CA in the next section. Then, in the following section, very simple self-reproducing CA are derived, also using the simulation of Turing machines.

SELF-REPRODUCTION: SIMPLE COMPUTATION, COMPLEX CA

In this section we will present a self-reproducing CA—a CA with a self-reproducing configuration—which is not necessarily simple, but is easy to describe. A Turing machine computation is described which does the desired thing. This Turing machine is wired in a CA which is then self-reproducing. The CA is, thus, as complex as required by the Turing machine that is wired in. In other words, the wired-in Turing machine "does everything." But first, just for exercise, we use the technique to prove self-reproduction where we relax the nontriviality requirement. We call this trivial self-reproduction *self-replication*.

[4]See the last section and last footnote for improvements to this.

SELF-REPLICATION, OR TRIVIAL SELF-REPRODUCTION

Wire in machine M which duplicates its input tape, repositions its head, duplicates all tape to the right of this position, and then repeats these steps forever. This can be represented as follows:

$$\uparrow x \rightarrow_M (x, \uparrow x) \rightarrow_M (x, x, \uparrow x) \rightarrow_M \cdots \tag{4}$$

where the vertical arrow indicates the position of the head after the computation indicated by the horizontal arrow immediately to its left.

Note that any one-dimensional configuration can be made to self-replicate in this scheme. Just embed a "head" cell at the left end of a given "tape" configuration; program the head cell to its initial state. Thus, we have proved the following theorem:

THEOREM 1 Let S be the set of finite one-dimensional configurations on finite state set Q. Then there exists CA Z_M in which s is a self-reproducing configuration, for all $s \in S$.

Since this result can be readily generalized to the d-dimensional case, we have answered in the affirmative a question raised by Lieblein[15]: Can any configuration be made to self-reproduce? Closely related to the theorem is the work of Waksman.[37] He has shown that a one-dimensional string can be made to self-replicate in a one-dimensional CA specially designed for the given string. Self-replication in the Waksman scheme proceeds in real time except for an initial "setting-up" time. "Realtime" here means that a string of length n is reproduced in n time steps. Note that our self-replication scheme is slower than real time (although linear time in n), but one CA serves for all strings.

NONTRIVIAL SELF-REPRODUCTION

Wire in universal Turing machine U' such that

$$\uparrow e'(P,x) \rightarrow_{U'} (e'(P,x), y, \uparrow e'(P,x)) \rightarrow_{U'} (e'(P,x), y, e'(P,x), y, \uparrow e'(P,x))$$
$$\rightarrow_{U'} \cdots$$

where e' is the encoding of programs and tapes required by U' and $x \rightarrow_P y$. Thus, U' first makes a copy of its input string. This can be done in such a way that the head of the universal machine never moves left of its initial position (at the leftmost symbol of its initial input string). Then, it simulates the computation of program P on its input x. If this computation halts with output y, then U' writes y on its output to the right of the copy of the input string $e(P,x)$. It is well known to Turing machine programmers that this simulation can be made to happen in the half-infinite space to the right of the input string copy so as not to destroy it during the simulation. Then another copy of the input string $e(P,x)$ is written to the right of y. Finally, the machine repositions its head to the leftmost symbol of the rightmost copy of string $e(P, x)$ and reenters its start state. Since U' is designed to ignore everything to the left of its start state position, the infinite computation represented above is achieved. Hence:

THEOREM 2 For any Turing machine P, there exists a CA $Z_{U'}$, and configuration c such that c self-reproduces and simulates P.

Note that P could be universal if desired. Hence:

Corollary 2.1. There exists a CA $Z_{U'}$, and configuration c such that c self-reproduces and is universal.

This corollary establishes the existence of universal (meaning computation universal) self-reproducing machines, but we are now also interested in obtaining the simplest such machines (in terms of state count per cell). Notice that U' above has to do a lot more than just simulation of an arbitrary computation. In particular, its restriction to half-infinite tapes will increase its complexity.

SELF-REPRODUCTION: COMPLEX COMPUTATION, SIMPLE CA

In this section we derive the main result of this paper. As opposed to the section above, the self-reproducing configuration "does everything." That is, for simplicity of the simulating CA, the wired-in computer U does as little as possible—i.e., nothing more is required of U than it be universal. The (4, 7) universal machine suggests to us just how simple (in terms of state and symbol count) U can be. In fact, since the simplest (decodeless) universal program U does just the following:

$$e(P',\ x') \rightarrow {}_U y'$$

where e is the encoding function required by U and

$$x' \rightarrow {}_{P'} y' ,$$

then a program P is desired such that

$$\uparrow x \rightarrow {}_P(e(P,x), y, \uparrow e(P,x)) \rightarrow {}_P(e(P,x), y, e(P,x), y, \uparrow e(P,x)) \rightarrow {}_P \ldots$$

where y is the result of some computation on x. For example, y might be a string of d background (blank or quiescent) symbols, b^d, so that $e(P, x)$ is "separated" from the second $e(P, x)$.

Note that if such a program P exists, then self-reproduction is straightforward. Simply take CA Z_U with U wired in and embed initial subconfiguration $e(P, x)$. Then the following situation holds:

$$\uparrow e(P,x) \rightarrow {}_U(e(P,x), y, \uparrow e(P,x)) \rightarrow {}_U(e(P,x), y, e(P,x), y, \uparrow e(P,x)) \rightarrow {}_U \ldots .$$

That is, the configuration representing $e(P, x)$ is self-reproducing. We now show that the desired program P exists and, furthermore, that it can be chosen to be universal. This result should be compared to a similar, but "one-shot," result on self-describing machines obtained by Lee.[14] He proved that there is a program P such that

$$x \rightarrow {}_P e(P, x)$$

with x the blank tape. Thatcher[32] created an actual example of such a machine.

LEMMA 3 For an arbitrary birecursive encoding function e from (program, tape) pairs to tapes and for an arbitrary partial recursive function g, with $g(x) = y$, there exists a self-describing machine with program P such that

$$\uparrow x \rightarrow {}_P(e(P,x), y, e(P,x)) \rightarrow {}_P(e(P,x), y, \uparrow e(P,x)) \rightarrow {}_P \cdots .$$

PROOF Define function h from programs into programs such that

$$\uparrow x \rightarrow {}_{h(Q)}(e(Q,x), y, \uparrow e(Q,x)) \qquad (a)$$

and

$$\uparrow e(Q,x) \rightarrow {}_{h(Q)}(e(Q,x), y, \uparrow e(Q,x)) . \qquad (b)$$

That is, $h(Q)$, for arbitrary program Q, is a program which first checks the input tape for a string of form $e(Q, x)$, for arbitrary x. This it can do because both e and Q are given it. If the input tape is not of form $e(Q, x)$, then $h(Q)$ executes the action indicated in (a) and described in temporal order below. As above, all computation is carried out in the singly infinite tape to the right of the initial head position which is the leftmost symbol of the initial tape.

1. $h(Q)$ first encodes the given program Q and input x with the given encoding function e to obtain $e(Q, x)$ and writes this on the tape while destroying the initial tape x.

2. It computes the given function g on argument x and writes the result y, if it is defined, to the right of the code $e(Q, x)$.

3. It writes the code $e(Q, x)$ again to the right of string y.

4. It repositions its head to the leftmost symbol of the rightmost code string $e(Q, x)$ and halts.

If the initial tape is of form $e(Q, x)$, then $h(Q)$ skips step (1) above and proceeds immediately to steps (2)–(4).

h is a total recursive function. Q and $h(Q)$ may not halt (see step 2 above), but $h(Q)$ is defined for every Q. Hence, by the Recursion Theorem, there exists a fixed-point program P for h—i.e., $h(P) = P$. For this program, (a) and (b) above become respectively

$$\uparrow x \rightarrow {}_{h(P)=P}(e(P,x), y, \uparrow e(P,x))$$

and

$$\uparrow e(P,x) \rightarrow_P (e(P,x), y, \uparrow e(P,x))$$

where $y = g(x)$ is defined. The lemma follows immediately if the final (halt) state of P is identified with its initial state. Clearly, if the halting version of P exists in the list of all programs, then the nonhalting version must also though, of course, with a different name or index.

THEOREM 4 Let Z_U be a universal CA with decodeless universal Turing machine U wired in. Then there exists a configuration c in Z_U which is self-reproducing and computes an arbitrary given partial recursive function g.

COROLLARY 4.1 There exists a CA Z_U and configuration c such that c self-reproduces and is universal.

PROOF Choose g in the theorem to be a universal Turing machine function.

I had hoped to prove the stronger result with the word "decodeless" omitted from the theorem above (and in fact claimed to have done so until my error was pointed out by Kristian Lindgren and Mats Nordahl). Then the universal CA of Smith[26] would have immediately implied the existence of one-dimensional self-reproducing universal machines with 2, 3, 4, 5, and 7 states per cell, for example.[5] These universal CA, however, are derived from Minsky's simplest known universal Turing machines which are not decodeless. I claim without proof that there are extremely simple decodeless universal Turing machines, perhaps adapted from Minsky's with the addition of one or two states, but leave it to some reader to find such a machine. It is only necessary that a universal machine be extended to decode its own output. This should be far simpler than the extensions beyond universality required by U' in the preceding section. Then the simplest known non-trivial self-reproducing machine immediately follows from the theorem and corollary above.

In comparison, von Neumann exhibited a 29-state universal CA capable (in two-dimensional) of supporting self-reproduction. His neighborhood—the so-called von Neumann neighborhood—consists of the five nearest-neighbor cells, including the cell itself. His construction is very lengthy and complex. Codd[10] was able to reduce the state count to 8 states per cell but his construction is also long and even more complicated. Codd's construction uses the so-called Moore neighborhood consisting of the 9 nearest-neighbor cells in two dimensions. Then Banks[6] created

[5]The complete selection of m-state $\times$ n-neighbor one-dimensional universal CA is 2×21, 3×13, 4×9, 5×8, 7×6, 11×4, 18×3, and 40×2.

the simplest known two-dimensional self-reproducing CA with 4 states and the von Neumann neighborhood.[6] By greatly increasing cell complexity,[3] Arbib was able to describe the processes simply. Here we have demonstrated both simple CA and simple descriptions by deriving CA with low-state count and using recursive function theory for compact constructions in them. Whereas, the earlier work in this area was confined to two dimensions, the results here are most elegant in one dimension although applicable to two or more dimensions. And finally, it is also striking that nowhere have we had to define "construction" in a CA to obtain self-reproduction—in fact, computation universality has been shown sufficient. The notion of "construction universality" used in earlier proofs is not required here.

Several comments about the class of self-reproducing machines developed here are in order. First, reproducing schemes based on the Recursion Theorem have been mentioned in several other places.[21,23,33] However, in all these cases the reproduction is one-shot—the process ceases after production of one offspring. The scheme of Theorem 4 produces offspring ad infinitum.

Second, if a self-reproducing configuration is interpreted to be "alive" only if it can reproduce again and again, then the machines of Theorem 4 are not alive. Once one of these machines computes and reproduces a single time, it becomes inactive. Of course, there is "no room" for another reproduction, but we will ignore this (degenerate) Malthusian dilemma and present a method for keeping a parent alive.

What we want is an augmentation of the scheme in the proof of Theorem 4 such that the head cell (active site) is also reproduced. This is how to accomplish the desired task:

1. A special symbol # is written after each string y, if y is defined.
2. When this situation occurs

$$\ldots, \#, e(P,x), y, \#, \uparrow e(P,x)\,,$$

   the cell representing # has the head cell in its neighborhood. When the head cell goes into its initial state H, the transition function f of the CA takes note of the juncture of # and H and creates a new head cell (indicated by an arrow pointing down):

$$\ldots, \#, e(P,x), y, \#, \uparrow e(P,x) \rightarrow \ldots, \#, e(P,x), y, \downarrow \#, \uparrow e(P,x)\,.$$

[6]Banks[6] also demonstrated a 2-state, Moore neighborhood (two-dimensional) universal CA and a 3-state, von Neumann neighborhood universal CA. More attention has been paid to another 2-state, Moore neighborhood (two-dimensional) CA which is universal.[8,36] It is the so-called game of Life popularized in the *Scientific American* in 1970 and 1971.[11] Codd[10] proves that no 2-state, von Neumann neighborhood universal CA can exist (hence, no $k = 2$, $r = 1$, Class 4 CA, in the terminology of Wolfram,[39] can be universal without changing the definition of universality (as done in Lindgren,[16] see last footnote)). In any event, none of these CA are known to support self-reproduction.

3. The new head cell propagates to the left until it encounters the next # to the left at which time it is in a position to reactivate the parent, and it does so by going into state H.

This scheme has the attraction of being a fully functional CA, with lots of computations proceeding in parallel, and not just a simulated Turing machine. (See Hurd[13] and Smith[29] for demonstrations of how CA exceed Turing machines.)

If other dimensions are available, then in step (3) the new head can, while propagating left, move the entire string between consecutive # states up one unit in another dimension. This, however, still leads to a Malthusian overcrowding (see Moore[20]—his "population" theorem goes through for our generalized cellular automata). Löfgren[17] gives a partial solution to this problem in terms of "birth" and "death" rates and "complexity," or number of cells per self-reproducing configuration. He does not, however, give the mechanics of self-reproduction assumed in his theory—each machine is known only to reproduce and be erased at certain rates.

The third and most important comment on our self-reproduction scheme is concerned with its biological relevance. Have we designed machines which actually *construct* other machines, or are the procedures introduced here just fancy copying routines somehow distinct from construction? Even with the modifications suggested in the second comment above, the model is still very non-biological (as are all the other tessellation models) in one obvious way—it is not parallel but serial in the extreme. Also the concept of growth is ignored in the sense that in our model a full-grown multi-cell offspring is constructed by the parent—not an undeveloped single-cell "egg" which grows under its own control into a replica of its parent.

Arbib[4] has exhibited an example of morphallactic regeneration of an elementary "worm" which proceeds in a highly parallel fashion. The worm is a one-dimensional string of identical cells divided into equal sections of head, body, and tail states. If a length of the worm is destroyed, the remaining cells reapportion themselves into equal thirds of head, body, and tail. This resembles the morphallaxis of hydra in the biological world. (A simple exercise for the interested reader is to embed a worm like that of Arbib in a one-dimensional CA designed so that the worm grows its missing parts in the correct proportion.[29]) Similarly, Lieblein[15] has detailed the CA realization of one of the highly parallel tessellation-like models of biological phenomena[7] such as reproduction, evolution, mutation, crossing-over, and even creation. Vitányi[34] added sexuality. Case[9] pursues self-describing computations which build distortions of themselves. My own closest approach to biological uses of the theory is Smith.[31]

TOTALISTIC SELF-REPLICATION AND SELF-REPRODUCTION

The recent revival of interest in CA has concentrated on the *totalistic* case, where the transition function is required to be an arithmetic function of the neighborhood states. A result that has been rediscovered numerous times is that trivial self-reproduction—that is, self-replication—is possible in totalistic CA.[2,18,22,38]

For nontrivial self-reproduction, the relevant result is due to Albert and Culik[1] who have discovered a one-dimensional universal CA that uses only the nearest neighbors for a neighborhood and 14 states per cell.[7] Inspection of their proof reveals that their CA has a background wave of activity which converts the quiescent state of the CA to a preparatory state. So there is a constantly expanding wavefront of multiple state changes to effect this preparatory state. Their universal computation operates in the midst of this "big bang" expanding wave. The proofs herein do not go through for this background activity. Nevertheless, with appropriate changes to the definitions—to allow an expanding background rather than a quiescent background—I believe the arguments could be made to go through. I have not attempted to do so.

[7] And shattered my record of 19 years for the simplest known one-dimensional universal CA with the 3 nearest-neighbor neighborhood! Mine had 18 states,[26] but the authors of Lindgren[16] show that a careful selection of state subset of my solution suffices for a 13-state solution. In fact, they do better than this: They show a new 9-state solution. (That is for an apples-to-apples comparison, for universal computation against an initially quiescent background. By relaxing the quiescent background requirement to permit periodic initial backgrounds, they are able to show a 7-state solution.) The most surprising result to me of the Albert-Culik work is that *totalistic* CA suffice for universality. Combining two of their results yields a 56-state, 3-neighbor, one-dimensional totalistic universal CA.

REFERENCES

1. Albert, Jürgen, and Karel Culik, II. "A Simple Universal Cellular Automaton and Its One-Way and Totalistic Version." *Complex Systems* **1** (1987): 1–16. Their CA simulates any other CA, a new form of CA universality. Their Theorem 1 is the one-dimensional case of Theorem 3.3 in Smith.[27]
2. Amoroso, Serafino, and Gerald Cooper. "Tessellation Structures for Reproduction of Arbitrary Patterns." *J. Comput. Sys. Sci.* **5** (1970): 455–464. There is an entire literature on tessellation automata, which are CA with programs—that is, the local transition function can change, in an SIMD way, at every step.
3. Arbib, Michael A. "Automata Theory and Development: Part I." *J. Theoret. Biol.* **14** (1967):131–156.
4. Arbib, Michael A. "Self-Reproducing Automata—Some Implications for Theoretical Biology." *Towards a Theoretical Biology, 2: Sketches*, edited by C. H. Waddington. Edinburgh University Press, 1969.
5. Arbib, Michael A. *Theoreis of Abstract Automata.* Prentice-Hall, 1969. The jacket of this textbook is my first cover art. It represents a self-reproducing machine!
6. Banks, E. Roger. "Cellular Automata." AI Memo No. 198, MIT Artificial Intelligence Lab, Cambridge, MA, 1970. His 2-state-9-neighbor, two-dimensional universal CA predates the universality proof of Life.
7. Barricelli, Nils Aall. "Numerical Testing of Evolution Theories. Part I: Theoretical Introduction and Basic Tests." *Acta Biotheoretica* (parts I/II) **16** (1962). This contains a large number of computer graphics of the type prominent in Wolfram,[39] which is surprising considering the crude computers available at the time.
8. Berlekamp, Elwyn R., John H. Conway, and Richard K. Guy. *Winning Ways for Your Mathematical Plays*, Vol. 2. New York: Academic Press, 1982. I have not read this reference, but it apparently contains Conway's proof of the universality of Life.
9. Case, John A. "Note on Degrees of Self-Describing Turing Machines." *J. ACM* **18** (1971): 329–338.
10. Codd, Edgar Frank. "Propagation, Computation and Construction in 2-Dimensional Cellular Spaces." Technical Report 06921-1-T, ORA, The University of Michigan, 1965. Also *Cellular Automata*, ACM Monograph Series. New York: Academic Press, 1968.
11. Gardner, Martin. "On Cellular Automata, Self-Reproduction, the Garden of Eden and the Game 'Life.'" *Sci. Amer.* **224** (1971): 112–117. I did the cover of this issue of the magazine. It is a finite CA recognizing the palindrome

TOOHOTTOHOOT in real time (see Smith[29]). See also Vol. 223, pp. 120–123, for Gardner's first Life article.

12. Holland, John H. "Universal Embedding Spaces for Automata." In *Cybernetics of the Nervous System*, edited by N. Wiener and J. P. Schade. Progress in Brain Research Series, Vol. 17. New York: Elsevier, 1965.
13. Hurd, Lyman P. "Formal Language Characterizations of Cellular Automata Limit Sets." *Complex Systems* **1** (1987): 69–80. Proves that one-dimensional CA can compute more than Turing machines! Presumably this is because of the CA ability to compute infinitely. See also Smith.[29]
14. Lee, Chester Y. "A Turing Machine Which Prints Its Own Code Script." *Mathematical Theory of Automata.* Microwave Research Institute Symposia Series, Vol. 12. Brooklyn, NY: Polytechnic Press, 1963.
15. Lieblein, Edward. "A Theory of Patterns in Two-Dimensional Tessellation Space." Ph.D. Dissertation, University of Pennsylvania, 1968. An underappreciated work, the first to show the regular set characterization of cellular automata patterns.
16. Lindgren, Kristian, and Mats G. Nordahl. "Universal Computation in Simple One-Dimensional Cellular Automata." *Complex Systems* **4** (1990): 299–318.
17. Löfgren, Lars. "Self-Repair as a Computability Concept in the Theory of Automata." *Proc. of the Symp. on Math. Theory of Automata.* Brooklyn, NY: Polytechnic Institute of Brooklyn, 1962.
18. Merzinich, Wolfgang. "Cellular Automata With Additive Local Transition." *Proceedings of the First International Symposium on Category Theory Applied to Computation and Control*, 186–194. Amherst, MA, Department of Computer and Information Sciences, University of Massachusetts, 1974.
19. Minsky, M. *Computation: Finite and Infinite Machines.* Englewood Cliffs, NJ: Prentice-Hall, 1969.
20. Moore, Edward F. "Machine Models of Self-Reproduction." *Proceedings of Symposia in Applied Mathematics, American Mathematical Society* **14** (1962):17–34.
21. Myhill, John. "The Abstract Theory of Self-Reproduction." In *Views on General Systems Theory*, edited by M. D. Mesarovic. John Wiley: New York, 1964.
22. Ostand, Thomas J. "Pattern Reproduction in Tessellation Automata of Arbitrary Dimension." *J. Comput. System Sci.* **5** (1971): 623–628.
23. Rogers, Hartley, Jr. *Theory of Recursive Functions and Effective Computability.* San Francisco: McGraw-Hill, 1967.
24. Smith, Alvy Ray. "Simple Computation-Universal Cellular Spaces and Self-Reproduction." *Proceedings of the 9th SWAT* (1968): 269–277. SWAT = IEEE Symposium on Switching and Automata Theory. At the 16th conference the

name changed to FOCS = IEEE Symposium on the Foundations of Computer Science. Although I have published several papers in this conference series (see also Smith[28]), I am most proud of my illustration which has been used by FOCS as its proceedings cover for the last 17 years, starting with the 14th conference.

25. Smith, Alvy Ray. "Cellular Automata Theory." Technical Report No. 2, Digital Systems Laboratory, Stanford University, Stanford, California, 1969. My Ph.D. dissertation. There are several bugs in this report, fixed in the publications below.
26. Smith, Alvy Ray. "Simple Computation-Universal Cellular Spaces." *J. ACM* **18** (1971): 339–353.
27. Smith, Alvy Ray. "Cellular Automata Complexity Trade-Offs." *Info. & Control* **18** (1971): 466–482. General CA theory.
28. Smith, Alvy Ray. "Two-Dimensional Formal Languages and Pattern Recognition by Cellular Automata." *Proceedings of the 12th SWAT* (1971): 144–152. A paper *Pattern Recognition by Finite Cellular Automata*, based on this work was accepted for publication by the *J. Comput. System Sci.* in 1975, but I never completed the submission since I changed to computer graphics about this time. Since then most of the results of this paper were published in *Picture Languages: Formal Models for Picture Recognition*, edited by Azriel Rosenfeld, Chapters 3 and 5. Academic Press, New York, 1979.
29. Smith, Alvy Ray. "Real-Time Language Recognition by One-Dimensional Cellular Automata." *J. Comput. System Sci.* **6** (1972): 233–253. Relationship of formal languages and one-dimensional CA.
30. Smith, Alvy Ray. "Introduction to and Survey of Polyautomata Theory." In *Automata, Languages, Development*, edited by Aristid Lindenmayer and Grzegorz Rozenberg, 405–422. Amsterdam and New York: North-Holland, 1976. This paper contains an extensive bibliography on CA. The book is the conference proceedings for what I propose be called the Artificial Life 0 conference, held in Noordwijkerhout, The Netherlands, April, 1975. Other participants were Karel Culik, Pauline Hogeweg, John Holland, Aristid Lindenmayer, and Stanislaw Ulam.
31. Smith, Alvy Ray. "Plants, Fractals, and Formal Languages." *Computer Graphics* **18** (1984): 1–10. I have called the L-systems or Lindenmayer systems used here dynamic cellular graph automata in the taxonomy of Smith.[30]
32. Thatcher, James W. "The Construction of a Self-Describing Turing Machine." In *Proceedings on the Symposium of the Mathamatical Theory of Automata*, 165–171. New York: Polytechnic Press, 1963.
33. Thatcher, James W. "Universality in the von Neumann Cellular Model." Technical Report 03105-30-T, ORA, The University of Michigan, 1964. Also

in *Essays on Cellular Automata*, edited by Arthur W. Burks, University of Illinois Press, 1970, a compendium of early papers.

34. Vitányi, Paul M. B. "Sexually Reproducing Cellular Automata." *Mathematical Biosciences* **18** (1973): 23–54.
35. von Neumann, John. *The Theory of Self-Reproducing Automata*, edited by Arthur W. Burks. Urbana: University of Illinois Press,1966. Von Neumann performed the work, completed here, in 1952-53.
36. Wainwright, Robert. "Life Is Universal." Proceedings of the Winter Simulation Conference, ACM, Washington, DC, 1974. I have not read this article but assume it is a presentation of the argument by R. William Gosper, circulated among Life cognoscenti who were the primary readers of Wainwright's Lifeline newsletter.
37. Waksman, Abraham. "A Model of Replication." *J. ACM* **16(1)** (1969).
38. Winograd, Terry A. "Simple Algorithm for Self-Replication." AI Memo No. 197, MIT Artificial Intelligence Lab, Cambridge, Massachusetts, 1970.
39. Wolfram, Stephen. *Theory and Applications of Cellular Automata*. Singapore: World Scientific, 1986. Represents well the combination of dynamic systems theory with CA, which revitalized the field in the 1980s.