Complex Systems 4 (1990) 299–318

Universal Computation in Simple One-Dimensional Cellular Automata

Kristian Lindgren
Mats G. Nordahl
Nordita, Blegdamsvej 17, DK-2100 Copenhagen, Denmark

Abstract. The existence of computation-universal one-dimensional cellular automata with seven states per cell for a transition function depending on the cell itself and its nearest neighbors ($r = 1$), and four states per cell for $r = 2$ (when next-nearest neighbors also are included), is shown. It is also demonstrated that a Turing machine with m tape symbols and n internal states can be simulated by a cellular automaton of range $r = 1$ with $m + n + 2$ states per cell.

1. Introduction

The history of cellular automata (CA) begins with the elaborate design of a two-dimensional automaton with 29 states per cell capable of universal computation and self-reproduction worked out by von Neumann in 1952–53 [1] (the details of the construction were completed by Burks and his student, Thatcher [1, 2]). The goal of the theory of automata envisaged by von Neumann was an understanding of the fundamental organizational principles of complex automata, natural and artificial. This led him to study two problems, that of reliable computation in the presence of noise [3] and that of the logical and organizational requirements for self-reproduction. He first considered a less abstract kinematic model of self-reproduction; the use of a cellular automaton was suggested by Ulam. The cellular automaton constructed by von Neumann was required to be both computation universal, to guarantee nontrivial behavior, and construction universal, to be capable of self-reproduction.

The von Neumann construction was simplified and improved in several respects by Codd [4]; in particular the transition function was made isotropic and the number of states per cell required was reduced to eight. This number was later reduced to four states per cell by Banks [5].

It is quite natural to treat the properties of computation universality and self-reproduction separately in this context, and in this article we shall only consider computation universality in cellular automata. If one only requires computation universality of a two-dimensional cellular automaton with the von Neumann neighborhood (the cell itself and its four nearest neighbors on a square lattice), it has been shown by Banks [5] that three states per cell are sufficient if the computation is required to be performed by a finite configuration in a blank background, and that two states per cell are sufficient if a periodic background is allowed. For a Moore neighborhood consisting of the cell and its eight nearest neighbors on a two-dimensional square lattice, two states per site are sufficient (even in a uniform background); the famous Game of Life is known to be computation universal [6].

Cellular automata in one spatial dimension have also been shown to be capable of universal computation. In the case of a transition function depending on the value of the cell itself and its right and left neighbors at the previous time-step (a CA rule of range $r = 1$), a computation-universal cellular automaton with 18 states per cell ($k = 18$) was constructed by Smith III [7], who also gave constructions with larger neighborhoods and fewer states per cell. These cellular automata were shown to be universal by constructing a simulation of a universal Turing machine. The number of states necessary for $r = 1$ was recently reduced to 14 by Albert and Culik II [8], whose universal cellular automaton was actually capable of simulating any other one-dimensional cellular automaton.

However, even extremely simple one-dimensional cellular automaton rules can show very complex behavior (e.g., [9, 10]). Wolfram has suggested that all cellular automaton rules belonging to class 4 in his phenomenological classification scheme [11] could be capable of universal computation [12]. Examples of such rules can be found with three states per cell for $r = 1$ and with two states per cell for $r = 2$. Embedding computation in these simple one-dimensional cellular automata would presumably require using propagating composite objects (particles) for the transfer of information (one might even speculate on the possibility of universal computation in $k = 2$, $r = 1$ CA rule 110, for which the existence of an infinite hierarchy of composite objects propagating with different velocities in a temporally and spatially periodic background is known [9, 13]). For these simple rules it is generally the case that such propagating objects are quite complex and hard to discover; furthermore, their collision properties cannot be tailored at will. This has lead some authors to instead consider so-called filter automata [14–16], which are updated sequentially (the rule for updating a site depends on states at the present time step on one side of the site and on states at the previous time step on the other). Such automata can in some cases support very large numbers of propagating structures, which may often display soliton-like behavior, meaning that particles retain their identities after collisions. This is clearly desirable if one wants to construct a more practical computational device than a Turing machine.

In this article we shall approach the problem of embedding computation in a one-dimensional cellular automaton from a different angle; instead of choosing a particular rule and then searching for useful propagating structures, we will design cellular automaton rules so that they support simple propagating structures. We then prove computation universality by designing the rule so that the collisions of these objects can be used in the simulation of a universal Turing machine. If in particular the universal Turing machine with four tape symbols and seven internal states constructed by Minsky [17] is used, we find that seven states per cell are sufficient for $r = 1$ and that four states per cell suffice for $r = 2$. These cellular automaton rules use a periodic background; we also give improved bounds on the number of states in the case of a uniform background.

This article is organized as follows. In section 2 some general formalism is introduced. Section 3 describes two ways of simulating an arbitrary Turing machine by a cellular automaton that lower known bounds on the number of states needed. Our main results are found in section 4, which contains the constructions of the small computation-universal cellular automata mentioned above. Section 5 contains a summary and discussion.

2. General formalism

We now consider a one-dimensional cellular automaton of range r, with variables σ_i at each site taking values in a finite set of symbols Σ, where $|\Sigma| = k$, and with dynamics given by a local transition rule $\phi : \Sigma^{2r+1} \to \Sigma$,

$$\sigma_i(t+1) = \phi(\sigma_{i-r}(t), \sigma_{i-r+1}(t), \ldots, \sigma_{i+r}(t)) \tag{2.1}$$

This local transition rule then induces a global cellular automaton mapping $\Phi : \Sigma^Z \to \Sigma^Z$ on a one-dimensional bi-infinite lattice.

A cellular automaton is said to be computation universal if it can compute any partial recursive function, which for example may be proven by showing that the cellular automaton can simulate a universal Turing machine. Various definitions of computation universality for a cellular automaton can be imagined, differing in technical details such as the class of initial states allowed and the allowed methods of detecting the end of a computation and interpreting the result. We shall in some cases use a notion of computation universality that differs slightly from that traditionally used, in that we allow the simulation to take place in a periodic background, possibly periodic both in space and time (this has sometimes been considered in the older literature as well [5, 18]). Traditionally, the existence of a unique quiescent state q (where $\phi(q, q, q) = q$) has been assumed, and the initial state has been required to have only a finite number of symbols different from q (e.g., [4, 7, 8]).

Our motivation for considering a more general notion of simulation is not only that we see little significant difference between starting from an infinite sequence of spatial period one (consisting of only quiescent states) and starting from a background of any other fixed period (which in both cases is then modified in a finite number of positions to produce the initial state), but more importantly that the simplest existing cellular automaton rules that are candidates for computation universality (such as $k = 2$, $r = 1$ CA rule 110, or $k = 3$, $r = 1$ totalistic rule 824 where the rules are numbered according to the conventions of Wolfram [19]) in fact do require a periodic background pattern for the propagation of particles. One may also note that if the more restricted notion of simulation in a quiescent background is adopted, Codd has shown that no computation-universal cellular automaton exists in two dimensions with $k = 2$ and a rotation symmetric five-cell neighborhood [4]. This rather ingenious proof based on signal propagation properties can be simplified to apply to the case of one dimension; one then finds that no one-dimensional $k = 2$, $r = 1$ cellular automaton rule can be computation universal. This result depends crucially on the assumption of a quiescent background (as mentioned in the introduction, Banks has constructed a two-dimensional $k = 2$ rule with the von Neumann neighborhood that is computation universal in a periodic background [5]); our point of view is to consider this an argument for abandoning this restriction.

The notion of simulation we will use can be described in a somewhat more formal way as follows. We assume that we are given a Turing machine M, which defines a partial recursive function $\Psi : T \rightarrow T$, where T is some countable set of Turing machine tapes, possibly required to have finite support (i.e., a finite number of nonblank symbols). The cellular automaton mapping $\Phi : \Sigma^Z \rightarrow \Sigma^Z$ is said to simulate the Turing machine M if there exist mappings $h : T \rightarrow \Sigma^Z$ and $g : \Sigma^Z \rightarrow T$, a finite string c of symbols in Σ, and a finite time N, so that $\Psi(t) = g(\Phi^N(h(t) \oplus c))$ whenever $\Psi(t)$ is defined. Here the operation $\oplus$ denotes either the insertion of, or the replacement of blanks by, a string $c \in \Sigma^*$ representing the Turing machine head and initial state. The functions g and h should obey some suitable restriction to ensure that the computation is done by the cellular automaton, not by the coding or encoding. Our examples are covered by assuming h to be an ϵ-free homomorphism (which maps each tape symbol to some nonempty string of symbols in Σ) and g to consist of an optional step where remnants of the head are removed (in some of our constructions the head erases itself at the end of a computation; this step is then unnecessary), followed by the inverse homomorphism h^{-1}. One should also require that the time N at which the computation ends can be detected through some simple operation. In our examples, the end of a computation is indicated by reaching a fixed point of the time evolution (or in one case by reaching a temporal cycle of length two). This could either be determined by comparing configurations at different time steps or by checking whether one of the finite subconfigurations representing the head is present.

The universal Turing machine that actually will be simulated in section 4 is the (4,7) (four tape symbols and seven internal states) machine constructed by Minsky [17]. The transition table of this Turing machine is given in table 1.

There are four tape symbols, y, 0, 1, and A, where 0 functions as the blank symbol, and seven internal states denoted 1...7. In section 4 we shall make use of the structure of this particular machine to minimize the number of cellular automaton states; we therefore note that not all combinations of out-states and left/right moves are allowed. In fact only 9 out of the 14

	y	0	1	A
1	$0L$	$0L$	$1L/2$	$1L$
2	$0L/1$	yR	AR	$yR/6$
3	yL	HALT	AL	$1L/4$
4	yL	$yR/5$	$1L/7$	$1L$
5	yR	$yL/3$	AR	$1R$
6	yR	$AL/3$	AR	$1R$
7	$0R$	$yR/6$	$1R$	$0R/2$

Table 1: The transition table of the (4,7) Turing machine constructed by Minsky. The table shows the symbol written, the move, and the resulting internal state, when a symbol σ is read in state q. If no new state is given, the machine remains in the same state.

conceivable combinations are allowed, namely $R/2$, $R/5$, $R/6$, $R/7$, $L/1$, $L/2$, $L/3$, $L/4$, and $L/7$.

3. Simulation of an arbitrary Turing machine

In this section, two ways of letting a one-dimensional cellular automaton simulate an arbitrary (m, n) Turing machine will be described. A construction by Smith III [7] accomplished this using $m + 2n$ states; we shall describe how this number may be reduced, either by using a periodic background or by making use of the detailed structure of the Turing machine being simulated.

The first of our constructions uses a periodic background, and simulates an arbitrary Turing machine using $k = m + n + 2$ states per cell — m states for the tape symbols σ_i, n states for the internal states of the Turing machine (symbolically denoted by q below), and two auxiliary states, denoted x and y below. The transition function of this cellular automaton (or rather the part of the transition function fixed by the simulation) is given in table 2; its operation during a right and a left move is shown below. A halting computation is handled by replacing the q representing the state by a symbol y. The cellular automaton then enters a temporal cycle of length two.

$$\begin{array}{cccccccc} x & \sigma_0 & x & q & \sigma & x & \sigma_1 & x \\ y & \sigma_0 & y & q' & \sigma' & y & \sigma_1 & y \\ x & \sigma_0 & q' & x & \sigma' & x & \sigma_1 & x \end{array} \qquad \begin{array}{cccccccc} x & \sigma_0 & x & q & \sigma & x & \sigma_1 & x \\ y & \sigma_0 & y & \sigma' & q' & y & \sigma_1 & y \\ x & \sigma_0 & x & \sigma' & x & q' & \sigma_1 & x \end{array}$$

$$\begin{array}{cccccccc} x & \sigma_0 & x & \sigma & q & x & \sigma_1 & x \\ y & \sigma_0 & y & q' & \sigma' & y & \sigma_1 & y \\ x & \sigma_0 & q' & x & \sigma' & x & \sigma_1 & x \end{array} \qquad \begin{array}{cccccccc} x & \sigma_0 & x & \sigma & q & x & \sigma_1 & x \\ y & \sigma_0 & y & \sigma' & q' & y & \sigma_1 & y \\ x & \sigma_0 & x & \sigma' & x & q' & \sigma_1 & x \end{array}$$

The two auxiliary symbols x and y serve as a modulo two clock, which means that one can have interactions taking place at even times and moves at odd times. This cellular automaton thus simulates an arbitrary Turing

$x\,\sigma\,x$	$\sigma\,x\,\sigma$	$\sigma\,x\,q$	$x\,q\,\sigma$	$q\,\sigma\,x$	$x\,\sigma\,q$	$\sigma\,q\,x$	$q\,x\,\sigma$	$\sigma\,x\,x$	$x\,x\,\sigma$
σ	y	y	$q'/\sigma'/y$	σ'/q'	q'/σ'	$\sigma'/q'/y$	y	y	y
$y\,\sigma\,y$	$\sigma\,y\,\sigma$	$\sigma\,y\,q$	$y\,q\,\sigma$	$q\,\sigma\,y$	$y\,\sigma\,q$	$\sigma\,q\,y$	$q\,y\,\sigma$	$\sigma\,y\,y$	$y\,y\,\sigma$
σ	x	q'	x	σ'	σ'	x	q'	x	x

Table 2: The transitions fixed by Turing machine simulation for the $k = m + n + 2$ universal cellular automaton. Here q and σ denote arbitrary internal states and tape symbols and q' and σ' denote the results of a Turing machine transition $(q, \sigma) \to (q', \sigma')$. When the transition function depends on the direction of the TM move, this is indicated by giving left/right(/halt) alternatives.

machine in twice linear time using $k = m + n + 2$ states per cell. If the (4,7) Turing machine discussed in section 2 is used (or the (5,6) almost-Turing machine constructed by Watanabe [20], which requires a periodic background; for a discussion of small Turing machines see [21]) one obtains a computation-universal cellular automaton with $k = 13$ and $r = 1$.

Universal cellular automaton rules with higher r can be obtained by coding the rule constructed above in different ways; for $r = 2$ one can let the tape symbols, x and y, remain intact and use a single additional symbol S. The seven internal states can then be represented as S followed by one of the seven symbols introduced. The cellular automaton rule constructed above can easily be adapted to this coding, which yields a universal $k = 7$, $r = 2$ rule. Other simple codes give universal cellular automata with $k = 5$ for $r = 3$, $k = 4$ for $r = 4$, $k = 3$ for $r = 5$, and $k = 2$ for $r = 9$. These results will be improved in our next section; we therefore leave out the details.

Another, somewhat simpler way of simulating an arbitrary Turing machine using an $r = 1$ cellular automaton is obtained by using a local symbol set Σ consisting of all tape symbols σ_i together with all allowed combinations of left/right moves and internal states of the Turing machine, here denoted $\overleftarrow{q_i}$ and $\overrightarrow{q_j}$ respectively (indices i and j run over the appropriate subsets of the internal states). This yields a total number of states k equal to $m +$ (number of allowed combinations of L/R moves and internal states) $\leq m + 2n$. This is essentially the simulation given by Smith III [7]; we have only added the observation that in general it is not necessary to use all $2n$ combinations of moves and states. This cellular automaton simulates the Turing machine in linear time; the details of the construction should be obvious from the two examples below (an initially left-moving head is treated similarly)

$$\begin{array}{ccccccc} \ldots & \sigma_0 & \sigma_1 & \overrightarrow{q} & \sigma_2 & \sigma_3 & \ldots \\ \ldots & \sigma_0 & \sigma_1 & \sigma_2' & \overrightarrow{q}' & \sigma_3 & \ldots \end{array} \qquad \begin{array}{ccccccc} \ldots & \sigma_0 & \sigma_1 & \overrightarrow{q} & \sigma_2 & \sigma_3 & \ldots \\ \ldots & \sigma_0 & \sigma_1 & \overleftarrow{q}' & \sigma_2' & \sigma_3 & \ldots \end{array}$$

If the (4,7) machine of Minsky's described in section 2 is used, we find that $k = 13$ is sufficient for universality for $r = 1$, since only 9 combinations of moves and internal states can occur. We may also note that this simulation takes place in the traditional quiescent background, with a quiescent state equal to the blank symbol. This cellular automaton rule can be coded in various natural ways for higher r; if, for example, the head is represented by two symbols instead of one, while the tape symbols are left intact, we obtain a universal $k = 7$, $r = 2$ cellular automaton, since the 9 possible pairs obtained using 3 symbols in addition to the tape symbols are sufficient to represent all move/state combinations. Other codings left as exercises for the reader give $k = 5$ for $r = 3$, $k = 4$ for $r = 4$, $k = 3$ for $r = 5$, and finally $k = 2$ for range $r = 9$. These are exactly the same values as obtained above; to improve on these we believe that one has to introduce composite objects to represent the internal states and/or the tape symbols, which is what we shall do next. For the rules described in this section, full information on the state transition (the tape symbol being read and the internal state) is available in the neighborhood of each site that is changed at the next time step as a result of the transition. This will no longer be the case in the following section.

4. Small universal cellular automata

In this section we shall attempt to reduce as far as possible the number of cellular automaton states needed for universal computation. In the preceding section, states and tape symbols of the Turing machine were represented by the values of single sites (for $r = 1$), which resulted in a minimal state count of $k = 13$. To improve this, we believe that composite objects have to be used, and our approach here will be to design the cellular automaton rule so that particles with collisions appropriate for the simulation of a universal Turing machine exist. Solving the constraints of having collisions suitable for the simulation of a certain Turing machine is a highly nontrivial task, which we shall attempt to simplify by using the smallest conceivable propagating objects.

For $r = 1$ we use a representation where the Turing machine head (and internal state) is represented by a left- or right-moving particle consisting of two symbols and where the tape symbols are represented by single stationary symbols separated by a number of blanks. The code used in the solution shown in table 3 represents an internal state as a particle of length two consisting of a tag symbol T indicating the direction in which the particle is moving and one additional symbol that determines the state. Since there are four right-moving and five left-moving state/move combinations, five different symbols in addition to the tag symbol are needed to represent the state; one of these will only occur in a left-moving combination. For these five symbols we will use the Turing machine tape symbols y, 0, 1, and A, together with one auxiliary symbol B. We have thus fixed part of the transition function by requiring yT, $0T$, $1T$, and AT to be right-moving particles in a background of blank symbols, by similarly requiring Ty, $T0$, $T1$, TA, and TB to be left-moving, and by requiring y, 0, 1, and A to be stationary in a blank

```
y T   y       y T   0       y T   1       y T   A
  y B y         y B 0         y 0 1         y A A
  B y 0         B 1 y         A y T         0 1 T
  T y A         A A y         A   y T       y   1 T
T y   0         0 0 A
                y y T
                y   y T
```

```
0 T   y       0 T   0       0 T   1       0 T   A
  0 B y         0 B 0         0 0 1         0 A A
  1 A 0         1 y y         y 0 T         B T T
  1 0 T         T 1 y         A   0 T       1   0 T
  B   0 T     T 1   y
  y     0 T
```

```
1 T   y         1 T   0     1 T   1       1 T   A
  1 B y           1 B 0       1 0 1         1 A A
  A A 0           A 0 y       B 1 T         1 1 T
  0 1 T           1 y A       A   1 T       1   1 T
  y   1 T         T 1 0
                T 1 B B
              T 1     A
```

```
A T   y       A T   0       A T   1       A T   A
  A B y         A B 0         A 0 1         A A A
  B A 0         B A y         1 A T         0 y T
  y A T         y A A         1   A T       0   y T
  0   A T       0 1 T
                y   1 T
```

```
  y   T y       0   T y       1   T y         A   T y
  y y y         0 y y         1 1 y           A 1 y
  y B y         0 0 y         1 0 y           0 A y
  B y 0         y 1 A         B B A           B y A
  T y A         T y A         y 1 1           T y 0
T y   0       T y   0         T 0 0         T y B A
                            T 0   1       T y     1
```

Figure 1: Continued next page.

```
    y   T 0
    y y 0
    y 1 A
   T y A
 T y   0
```

```
0   T 0
0 y 0
0 0 A
y y T
y   y T
```

```
1   T 0
1 1 0
1 0 B
B 1 B
A y T
A   y T
```

```
A   T 0
A 1 0
0 A B
B 0 y
y 0 A
A 1 T
0 1 1 T
y     1 T
```

```
    y   T 1
    y y 1
    y 1 y
   T 1 y
 T 1   y
```

```
0   T 1
0 y 1
0 T y
  0
```

```
    1   T 1
    1 1 1
    1 y 0
   T 1 A
 T 1   A
```

```
      A   T 1
      A 1 1
      0 0 0
      y A 1
      0 B A
      1 y 1
      T A y
    T A B A
  T A     1
```

```
    y   T A
    y y A
    y 1 0
   T A B
 T A   y
```

```
0   T A
0 y A
0 A 0
B 0 T
y   0 T
```

```
     1   T A
     1 1 A
     1 B A
     A B 1
     B y 1
     T B y
   T B 0 0
 T B     1
```

```
    A   T A
    A 1 A
    0 B A
    1 y 1
    T A y
  T A B A
T A     1
```

```
y   T B
y y B
y A T
0   A T
```

```
0   T B
0 y B
0 1 T
y   1 T
```

```
1   T B
1 1 B
1 A T
1   A T
```

```
A   T B
A 1 B
0 y T
0   y T
```

Figure 1: Cont'd. The results of collisions between particles and stationary objects for the universal $k = 7$, $r = 1$ cellular automaton. The collisions take place in a background of blank symbols; these are represented by empty positions in the figure.

background. It remains to show that the rest of the transition function can be chosen so that the constraints of having collisions between particles and stationary objects corresponding to table 1 can be solved.

One solution to these constraints is given in table 3, and the results of the 36 possible collisions between particles and stationary symbols for this cellular automaton rule are shown in figure 1. Here the right-moving particles yT, $0T$, $1T$, and AT represent internal states 2, 5, 6, and 7 of table 1, and the left-moving particles Ty, $T0$, $T1$, TA, and TB represent internal states 1, 2, 3, 4, and 7. This figure, together with a recipe for constructing initial states, decoding final states, and detecting a halt, constitutes our demonstration of computation universality for this cellular automaton.

The initial state for simulating the universal Turing machine is obtained by first inserting two blanks between each pair of symbols on the Turing machine tape. This is the minimal number of blanks needed to ensure that the collisions do not interfere. The Turing machine of table 1 is initially in state 2; to obtain the initial state for the cellular automaton we thus insert yT to the left (or $T0$ to the right) of the first symbol to be read. This string is inserted between the two blanks that separate the tape symbols. The cellular automaton rule is constructed so that tape symbols are moved two positions backwards when a Turing machine head passes through but remain stationary when the head changes its direction. This ensures that the distance between tape symbols is constant (equal to three sites), except when they are separated by a TM head, in which case the distance is increased by two sites.

The simulation is then carried out by iterating the cellular automaton mapping until a fixed point is reached, indicating the end of the computation. The resulting Turing machine tape is recovered by erasing all blanks (note that this is not quite the inverse homomorphism; when the head erases itself it leaves two extra blanks, which first have to be removed).

The time needed for this simulation is bounded above by $7N$, where N is the number of steps in the computation performed by the Turing machine. This upper limit follows from the length of the longest collision, together with the fact that the tape symbols are separated by two blanks.

The solution of table 3 is by no means unique; we have found a number of other solutions, but so far none where all collisions are of identical length (though this easily can be accomplished by introducing an additional symbol). One can also contemplate similar constructions starting from a blank background, where the blank symbols of the Turing machine and the cellular automaton are identified. For $r = 1$ we have obtained solutions with $k = 9$ in this case. This number could quite possibly be improved further.

We have also constructed a computation-universal cellular automaton with $k = 4$ and $r = 2$ using similar ideas. The transition table of this cellular automaton is given in table 4, and the elementary collisions are shown in figure 2. In this case we have right-moving particles 00+, 01+, 10+, and 11+ (in a blank background) representing internal states 2, 5, 6, and 7, and left-moving particles +00, +01, +10, +11, and ++0 representing internal states 1, 2, 3, 4, and 7. The tape symbols y, 0, 1, and A are represented by the strings 00, 01, 10, and 11. The initial state can be obtained, for example, by inserting three blanks between each pair of symbols on the Turing machine tape and then inserting the appropriate string (00+ or +01) for the head. The time required for the simulation is then bounded above by 5N.

This rule can easily be modified to operate in a uniformly blank background if one extra state is introduced; a particle moving into a blank region can then be designed to keep track of the distance it has moved and act as if 01 (the TM blank symbol) was encountered after three steps. This means that $k = 5$, $r = 2$ is sufficient for computation universality in a quiescent background.

We have not attempted to search for universal rules with higher r in the same manner, though in principle this could be done with more effort. Instead, we can give improved bounds on the number of states needed for universality for higher r by coding the rules constructed above in different ways. In this way we have obtained universal rules with $k = 3$ for $r = 4$ and with $k = 2$ for $r = 8$, which improves our results from section 3. A straightforward example of a construction that gives $k = 2$, $r = 8$ is given by coding the seven symbols of the $r = 1$ cellular automaton using the length six blocks given by 011 followed by 000, 001, 010, 100, 101, 110, or 111. The string 011 can then appear only at block boundaries, and a range of $r = 8$ is sufficient to determine all transitions. An example of a code for $r = 4$, $k = 3$ is given by the blocks $200 = \sqcup$, 201, 210, 211, 220, 221, $222 = T$. This example is a bit more subtle; it is necessary to make use of the detailed properties of the simulation, in particular the fact that the combination TT only can occur in the context $BTT\sqcup$, to see that block boundaries can be uniquely determined.

We have thus constructed universal one-dimensional cellular automata with $k = 7$ for $r = 1$, $k = 4$ for $r = 2$, $k = 3$ for $r = 4$, and $k = 2$ for $r = 8$. This can be compared to previously known universal one-dimensional cellular automaton rules, which have $k = 14$ for $r = 1$, $k = 4$ for $r = 4$, $k = 3$ for $r = 6$, and $k = 2$ for $r = 10$ [7, 8].

5. Summary and discussion

We have designed simple cellular automaton rules capable of universal computation. Universality was shown by simulating a small universal Turing machine. The simulation was first done in a straightforward way, where states and tape symbols of the Turing machine all corresponded to different cellular automaton states. To lower the number of cellular automaton states, we instead used a representation where the states and tape symbols were composite objects. The rule was then designed so that the collisions of these objects corresponded to the state transitions of the Turing machine. This allowed us to construct universal rules with $k = 7$ for $r = 1$ and $k = 4$ for $r = 2$.

It is quite conceivable that the idea of designing composite structures could be extended to demonstrations of universality for even simpler rules and to the design of other computational devices. The structures involved would then in general have to be more complex, and the discrete equations corresponding to the collision constraints would become more difficult to solve (this was done by hand for the cellular automaton rules of this article; we have also considered using methods such as simulated annealing to solve the constraints).

```
0 0 +   0 0
  0 0 0 0 0
  0 0 1 ␣ +
  + 0 0 ␣ 1
+ 0 0   0 1
```

```
0 0 +   0 1
  0 0 0 0 1
  0 0 0 ␣ 0
  0 ␣ 0 1 0
  0 0 0 0 +
  0 0   0 0 +
```

```
0 0 +   1 0
  0 0 0 1 0
  0 + 0 0 +
  1 1   0 0 +
```

```
0 0 +   1 1
  0 0 0 1 1
  0 + 0 ␣ +
  1 + 1 0 +
  0 0   1 0 +
```

```
0 1 +   0 0
  0 1 0 0 0
  0 ␣ 0 1 +
  0 0   0 1 +
```

```
  0 1 +   0 1
    0 1 0 0 1
    0 ␣ 1 1 0
    + 1 0 + 0
  + 1 + 0 0 1
+ 1 0     0 0
```

```
0 1 +   1 0
  0 1 0 1 0
  0 + 0 1 +
  1 1   0 1 +
```

```
0 1 +   1 1
  0 1 0 1 1
  0 + 0 + +
  1 0 0 1 +
  1 0   0 1 +
```

```
1 0 +   0 0
  1 0 0 0 0
  1 1 0 ␣ +
  1 + 1 0 +
  0 0   1 0 +
```

```
1 0 +   0 1
  1 0 0 0 1
  1 1 1 ␣ 0
  + 0 ␣ + +
+ 1 0   1 1
```

```
1 0 +   1 0
  1 0 0 1 0
  1 0 1 0 +
  1 1   1 0 +
```

```
1 0 +   1 1
  1 0 0 1 1
  1 0 0 ␣ +
  1 ␣ 1 0 +
  1 0   1 0 +
```

```
1 1 +   0 0
  1 1 0 0 0
  1 + 1 1 +
  0 1   1 1 +
```

```
1 1 +   0 1
  1 1 0 0 1
  1 + 0 1 0
  + ␣ 1 0 +
  0 0   1 0 +
```

```
1 1 +   1 0
  1 1 0 1 0
  1 ␣ 1 1 +
  1 0   1 1 +
```

```
1 1 +   1 1
  1 1 0 1 1
  1 ␣ 0 + +
  + ␣ 0 0 +
  0 1   0 0 +
```

```
  0 0   + 0 0
  0 0 1 0 0
  + 0 0 ␣ 1
+ 0 0   0 1
```

```
  0 1   + 0 0
  0 1 1 0 0
  0 0 1 ␣ 1
  + 0 0 + 0
+ 0 0   0 1
```

```
    1 0   + 0 0
    1 0 1 0 0
    1 1 1 ␣ 1
    + 0 0 0 0
  + 0 + ␣ ␣ +
+ 0 1     1 0
```

```
  1 1   + 0 0
  1 1 1 0 0
  + ␣ + ␣ 1
  + 0 0 ␣ 0
+ 0 0   1 0
```

Figure 2: Continued on next page.

```
  0 0   + 0 1   0 1   + 0 1     1 0   + 0 1     1 1   + 0 1
  0 0 1 0 1     0 1 1 0 1       1 0 1 0 1       1 1 1 0 1
  + 0 + + +     0 0 0 0 +       1 1 0 + +       + ␣ 1 0 +
+ 0 0   0 1     0 0   0 0 +     1 0 0 0 +       0 0   1 0 +
                                1 1   0 0 +
```

```
  0 0   + 1 0   0 1   + 1 0       1 0   + 1 0       1 1   + 1 0
  0 0 1 1 0     0 1 1 1 0         1 0 1 1 0         1 1 1 1 0
  + 1 0 ␣ 0     0 + + 1 0         1 + + ␣ 0         + 1 1 1 0
+ 1 0   0 0     0 1 ␣ ␣ ␣         + 1 0 ␣ 1       + 1 1   1 0
                                + 1 0   1 1
```

```
  0 0   + 1 1   0 1   + 1 1       1 0   + 1 1       1 1   + 1 1
  0 0 1 1 1     0 1 1 1 1         1 0 1 1 1         1 1 1 1 1
  + 1 1 ␣ 1     0 + + + 1         1 + 0 ␣ 1         + 1 1 + 1
+ 1 1   0 0     0 ␣ 0 1 +         + + 0 + 1       + 1 1   1 0
                0 0   0 1 +     + + 0   1 0
```

```
0 0   + + 0     0 1   + + 0     1 0   + + 0     1 1   + + 0
0 0 1 + 0       0 1 1 + 0       1 0 1 + 0       1 1 1 + 0
+ ␣ 1 1 +       0 1 1 0 +       1 ␣ 1 1 +       + ␣ 0 0 +
0 1   1 1 +     0 0   1 0 +     1 0   1 1 +     0 1   0 0 +
```

Figure 2: Cont'd. The results of collisions between particles and stationary objects for the universal cellular automaton rule of table 4. The collisions take place in a background of blank symbols; blanks are represented by empty positions in the initial and final state, and by ␣ in intermediate states.

It has often been remarked that even very simple cellular automaton rules can show very complex behavior. Even if a cellular automaton is known to be computation universal, this property refers only to a small subset of initial states and might have little bearing on the behavior for generic initial states (a particularly striking example of this is given by the universal von Neumann rule of Banks [5]). Simulations of our universal $r = 1$ and $r = 2$ rules for random initial states are shown in figures 3 and 4. The rules are only partially fixed by the Turing machine simulation; two natural choices for the remaining transitions are identity (i.e., to keep the value of the center symbol) or to introduce the blank symbol in all these transitions. The first of these conventions is used in figure 3, the second in figure 4. We find behavior typical of class 4 in Wolfram's classification [11].

	y	0	1	A	B	␣	T
yy	B	1	1	1	A	y	␣
$y0$			y	1		A	␣
$y1$	1	A	0	y		y	
yA			B	1		0	␣
yB	y	1		␣		T	
y␣	␣	␣	␣			␣	y
yT						y	
$0y$	0	0	T	A	1	A	␣
00	1	A	0	y		1	
01			␣			T	␣
$0A$	y	0		T	0	T	
$0B$	A	y		y		B	
0␣	␣		␣	␣		␣	y
$0T$	0					0	
$1y$	1	1	A	1		y	
10	B		1		1	B	␣
11	0	0	y	B	A	0	␣
$1A$		0		1		A	␣
$1B$	A	0		B	␣	T	
1␣	␣	␣	␣	␣		␣	1
$1T$						1	
Ay						A	␣
$A0$	y		A			T	
$A1$	A	A	0	B	y	A	1
AA	0	1		y		T	
AB	A	A	y	␣		y	
A␣	␣	␣	␣			␣	1
AT						A	
By		y	B	y		0	
$B0$	0	␣				y	␣
$B1$	A				y	1	␣
BA	A	A				1	
BB				1		A	
B␣		␣				␣	
BT							␣
␣y	y	A	T	0	B	y	␣
␣0	0	y	y	B	1	0	␣
␣1	T	B	1	1	A	1	␣
␣A	A	1	0	0	B	A	␣
␣B	T	y	A	y	y	y	1
␣␣	␣	␣	␣	␣	␣	␣	T
␣T	y	0	1	A	B		
Ty		B		␣	␣	␣	
$T0$		␣				␣	
$T1$	␣	B		␣	␣	␣	
TA	B				␣	␣	
TB	0	␣				␣	
T␣	B	B	0	A		T	
TT						0	

Table 3: The transition function of the computation-universal $k = 7$, $r = 1$ cellular automaton. The blank symbol is represented by ␣ ; empty positions indicate transitions not fixed by the simulation.

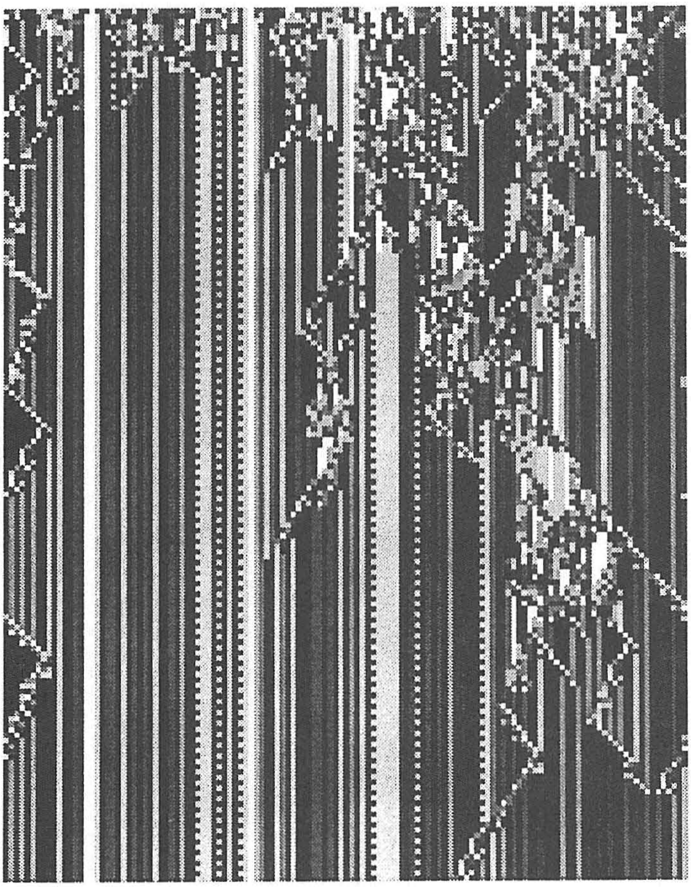

Figure 3: A simulation of the $k = 7$, $r = 1$ universal CA of table 3 for an uncorrelated initial state (with a density of blanks equal to 0.76). Symbols y, 0, 1, A, B, $\sqcup$, and T are represented by ▓░▒▓░■□

We have also simulated rules where the free positions in the rule table were assigned random values. For the case of $r = 1$, where 212 out of 343 positions are determined by the Turing machine simulation, a subjective classification of 500 randomly chosen rules gave 489 instances of typical class-4 behavior (reminiscent of figure 3) and 11 whose behavior could be described as spatiotemporal intermittency (e.g., [22]), i.e., coexistence of slowly evolving regular and disordered domains.

For the $r = 2$ case, where only 434 out of 1024 positions are determined by the simulation, 500 randomly chosen rules gave 66 cases with class-4

	00	01	0␣	0+	10	11	1␣	1+	␣0	␣1	␣␣	␣+	+0	+1	+␣	++
000	1	0	␣	␣	0	0	␣		0		+				0	
001	0	+	0		0	1	␣			0	0	0	1		0	
00␣	␣	␣	1		␣		0		␣	␣	␣	␣	1	1	0	1
00+			0						0	0	0					
010	0	1	␣		0	0	+				+				1	
011	1	0	␣	␣	+	+	␣				+		1			
01␣	␣					␣	+		␣	␣	␣	␣	1	1	␣	1
01+			1						1	1	1					
0␣0			0	␣	0		0	␣			0					
0␣1			1	␣	0		1	␣			1					
0␣␣	␣	␣			␣	␣			␣	␣	␣	␣	+	+		+
0␣+	0	0			1	1					+		+		1	
0+0				␣				␣			1	1				0
0+1											0					
0+␣	0	0			0	0			+	+	+	␣				
0++					␣						+			0	0	
100	0	1	1	␣	1	0	1	␣			1	1	1			
101	1	0	1	␣	+	0	+				+		1			
10␣	␣	␣	0		␣	␣	1		␣	␣	␣	␣	1	1	0	1
10+			0						0	0	0				0	
110	1	0	␣		1	0	0				0	1			1	0
111	+	1	1		1	1	+		␣	0	1		0			
11␣	␣	␣	+		␣		0		␣	␣	␣	␣	1	1		1
11+			0				1		1	1	1					
1␣0			0	␣				␣			+					0
1␣1			1	␣				␣			0					
1␣␣	␣	␣			␣	␣			␣	␣	␣	␣	+	+		+
1␣+	0	0			1	1					1		+			
1+0		␣			1					0	+					
1+1				␣				␣			0					
1+␣	0	0			0	0			+	+	+					
1++									0							
␣00	0	+	␣		0	1	0	␣	0	0	0	0			0	
␣01	␣	+	0		0	+		1	1	1	1	1			0	
␣0␣		0				1					␣					
␣0+	1	1	+	0										1	0	␣

Table 4: Continued on next page.

	00	01	0␣	0+	10	11	1␣	1+	␣0	␣1	␣␣	␣+	+0	+1	+␣	++
␣10	1	0	␣		1	+		␣	0	0	0	0			1	
␣11	+	␣	+	0	␣	1	0	␣	1	1	1	1			1	
␣1␣				␣	0	0					␣					
␣1+		␣	+		0	1									1	
␣␣0	0	+	0	␣	0	0	0	␣	0	+			1			0
␣␣1	1	1	1	␣	1	+	1	␣	+	1			+	0		+
␣␣␣	␣	␣	␣	␣	␣	␣	␣	␣	␣	␣	␣	␣	+	+	␣	+
␣␣+	0	0	1	0	1	1		1	0	0	0	+	+			
␣+0	+		0	0			1	␣				0			1	0
␣+1			0	+		1	1	1					0			
␣+␣	1				0	1	␣				␣				0	
␣++			0	0							1					
+00	␣						0		␣	␣	␣		␣		0	
+01			0								␣				0	
+0␣							+				␣				0	␣
+0+							1				␣				1	␣
+10									␣	␣	␣		0		1	
+11					␣				␣	␣	␣			␣	1	
+1␣											␣					
+1+	␣															
+␣0			0	␣			0				1					
+␣1			1	␣			1	␣			0					
+␣␣	␣	␣			␣	␣			␣	␣	␣				1	
+␣+										0						
++0										␣	␣			␣		
++1			␣								+					
++␣			␣								␣					
+++							1				1					

Table 4: Cont'd. The transition function of the universal $k = 4$, $r = 2$ cellular automaton. The blank symbol is represented by ␣; empty positions indicate transitions not fixed by the simulation.

behavior, and 434 cases classified as spatiotemporal intermittency or class 3. This clearly shows the effect of having a less constrained rule table.

The tendency toward class-4 behavior when a larger part of the rule table is fixed by the simulation illustrates the difference between one and two spatial dimensions. For the one-dimensional universal cellular automata discussed in this article, signals are carried by particles propagating in a constant or periodic background; when most of the rule is fixed by the description of their propagation and collisions, it seems likely that asymptotically one will find a decreasing density of particles in this background. Coupled to more irregular transient behavior, this would yield behavior reminiscent of $k = 2$,

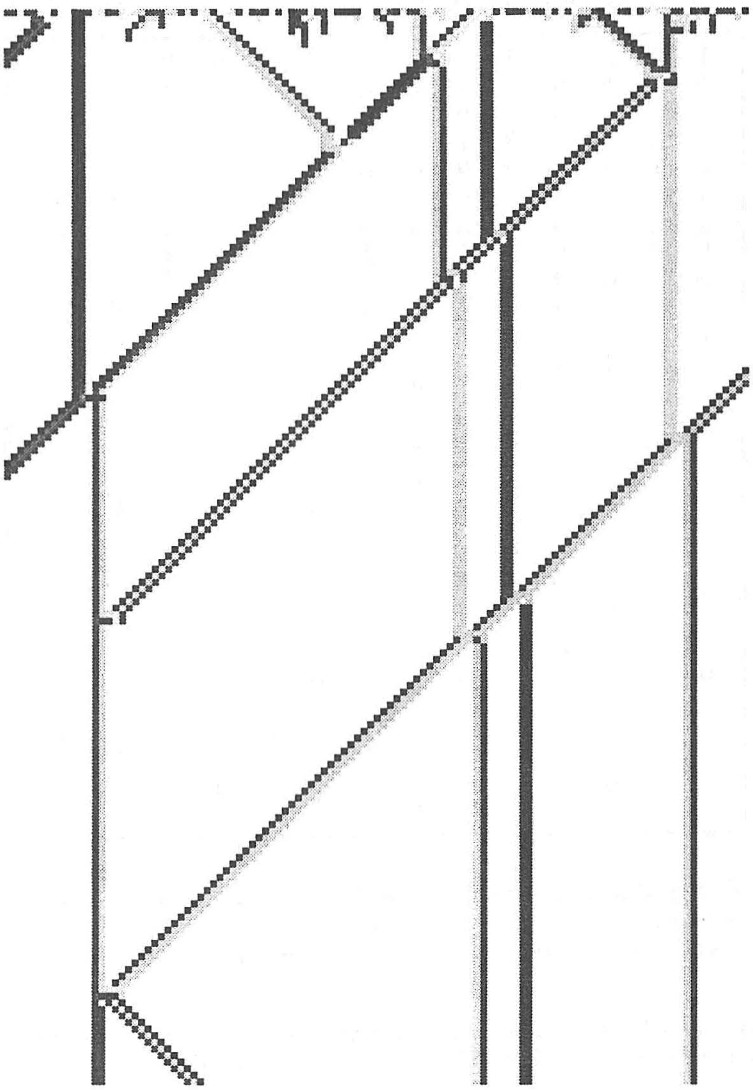

Figure 4: The $k = 4$, $r = 2$ universal cellular automaton of table 4 simulated starting from a random initial state. The symbols 0, 1, ⊔, and + are represented by ▩ ▨ □ ■

$r = 1$ CA rule 110, which could be classified as class 4. This argument is only sensible if the proper background emerges from random initial states; this could be made plausible through mean-field theory or higher-order local structure theory [23]; here we only note (in an approximation of zeroth order) that the blank symbol is the most frequent in table 3 and that a blank background is stable.

In two dimensions, signals can be restricted to propagate on wires without any significant increase of the number of states (e.g., [5]), which makes the spontaneous emergence of a proper background for signal propagation unlikely. No correlation between class-4 behavior and provable computation universality can be expected in this case. On the other hand, for the Game of Life, where the signals (gliders) used in proving universality [6] propagate in a blank background, class-4 behavior is observed.

One may finally note that the behavior of the universal rules of figures 3 and 4 is in some sense still simple, in that the composite structures used are either stationary or propagate with one particular speed, which allows us to establish the correspondence with a Turing machine. It is an open question how to characterize the behavioral complexity of rules with more complex sets of particles where collisions occur in a more unpredictable way.

Acknowledgments

We are grateful to Doyne Farmer for pointing out a reference and to Stephen Wolfram for comments on the manuscript. This research was supported by the Erna and Victor Hasselblad Foundation.

References

[1] J. von Neumann, *Theory of Self-Reproducing Automata*, edited and completed by A.W. Burks (Univ. of Illinois Press, 1966).

[2] A.W. Burks, *Essays on Cellular Automata*, (Univ. of Illinois Press, 1970).

[3] J. von Neumann, "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," *Collected Works*, vol. V (Macmillan, New York, 1963) 5.329–5.378.

[4] E.F. Codd, *Cellular Automata* (ACM Monograph Series, Academic Press, 1968).

[5] E.R. Banks, *Information Processing and Transmission in Cellular Automata*, thesis, Massachusetts Institute of Technology, 1971.

[6] E.R. Berlekamp, J.H. Conway, and R.K. Guy, *Winning Ways for Your Mathematical Plays*, Vol. 2: *Games in Particular* (Academic Press, New York, 1982).

[7] A.R. Smith III, "Simple computation-universal cellular spaces," *Journal ACM*, **18** (1971) 339–353.

[8] J. Albert and K. Culik II, "A simple universal cellular automaton and its one-way and totalistic versions," *Complex Systems*, **1** (1987) 1–16.

[9] S. Wolfram, ed., *Theory and Applications of Cellular Automata* (World Scientific, Singapore, 1986).

[10] K. Lindgren and M.G. Nordahl, "Complexity measures and cellular automata," *Complex Systems*, **2** (1988) 409–440.

[11] S. Wolfram, "Universality and complexity in cellular automata," *Physica*, **10D** (1984) 1–35.

[12] S. Wolfram, "Twenty problems in the theory of cellular automata," *Physica Scripta*, **T9** (1985) 170–185.

[13] K. Lindgren, in preparation.

[14] J.K. Park, K. Steiglitz, and W.P. Thurston, "Soliton-like behavior in automata," *Physica*, **D19** (1986) 423–432.

[15] K. Steiglitz, I. Kamal, and A. Watson, "Embedding computation in one-dimensional automata by phase coding solitons," *IEEE Transactions on Computers*, **C-37** (1988) 138–145.

[16] C.H. Goldberg, "Parity filter automata," *Complex Systems*, **2** (1988) 91–141.

[17] M. Minsky, *Computation: Finite and infinite machines*, (Prentice Hall, Englewood Cliffs, NJ, 1967).

[18] C.Y. Lee, "Synthesis of a cellular computer," in *Applied Automata Theory*, J.T. Tau, ed. (Academic Press, 1968).

[19] S. Wolfram, "Statistical mechanics of cellular automata," *Reviews of Modern Physics*, **55** (1983) 601–644.

[20] S. Watanabe, "5-symbol 8-state and 5-symbol 6-state Turing Machines," *Journal ACM*, **8** (1961) 476–484.

[21] L. Priese, "Towards a precise characterization of the complexity of universal and nonuniversal Turing Machines," *SIAM Journal of Computing*, **8** (1979) 508–523.

[22] H. Chaté and P. Manneville, "Coupled map lattices as cellular automata," *Journal of Statistical Physics*, **56** (1989) 357–370.

[23] H.A. Gutowitz, J.D. Victor, and B.W. Knight, "Local structure theory for cellular automata," *Physica*, **D28** (1987) 18–48.