EMBEDDING COMPUTATION IN ONE-DIMENSIONAL AUTOMATA BY PHASE CODING SOLITONS

Kenneth Steiglitz
Irfan Kamal
Arthur Watson

CS-TR-015-86

November, 1985

Embedding Computation in One-Dimensional Automata by Phase Coding Solitons[†]

Kenneth Steiglitz, Irfan Kamal, Arthur Watson,
Dept. of Computer Science
Princeton University
Princeton, NJ 08544

Abstract

We show that some kind of meaningful computation can be embedded in very simple, microscopically homogeneous, one-dimensional automata — filter automata with a parity next-state rule.

A systematic procedure is given for generating moving, periodic structures ("particles"). These particles exhibit soliton-like properties; that is, they often pass through one another with phase shifts. We then discuss ways to encode information in the phase of these particles.

Finally, the search for useful logical operations is reduced to a search for paths in certain graphs. As a demonstration of principle, we give the details of implementing a carry-ripple adder.

1. Introduction

In [1] a class of one-dimensional automata was described, called the *parity-rule filter automata* (parity-rule FA's), which support particles with soliton-like properties. That is, although the operation of the automaton is nonlinear and irreversible, moving persistent structures pass through one another while preserving their identities. In this paper we will study the systematic generation and properties of these solitons. We will then show how they can be used to encode information and perform useful computation — using the carry-ripple adder to demonstrate the principle.

Embedding computation in a simple one-dimensional automaton has important practical advantages over the two-dimensional alternative. One-dimensional structures are easier to build and operate, they can be looped naturally, and no a priori decisions need be made about the size of the array. A one-dimensional automaton can be implemented in a highly parallel way much more easily than can a two-dimensional one, because the cell values can be streamed serially through many cascaded processors. In [2], for example, a VLSI implementation of a simple, fixed r=2 CA was described which performs more than 10^8 updates per second per chip. The concatenation of many identical processors then results in a highly concurrent machine with a fixed, regular structure.

The principal motivation of this work is the exploration of microscopically homogeneous structures that support computation. Pseudo-particles then play a

[†] This work was supported in part by NSF Grant ECS-8414674, U. S. Army Research-Durham Contract DAAG29-85-K-0191, DARPA Contract N00014-82-K-0549, and ONR Grant N00014-83-K-0275.

natural role in providing a medium for transmitting information from one place to another. As we will see, solitons are especially interesting pseudo-particles from this point of view, because they bear information in their carrier and envelope phase, and this information is changed when solitons collide.

Carter [3] discusses the possibility of implementing logical functions to build cellular automata (CA's) using true (physical) solitons supported by chemical structures, while we discuss the opposite: implementing logical functions using the soliton-like structures that arise in certain automata. A further practical aspect of our work is the possibility that we can use true solitons in much the same way that we use those supported by automata.

2. Filter Automata

For the purposes of this paper we will restrict discussion to the special class of one-dimensional, binary-state, filter automata described in [1]. The site values will be denoted by a_i^t, $i = 1,...,n$, where the superscript is the *time* variable t, $0 \leq t \leq +\infty$, and the subscript is the *space* variable i, $-\infty \leq i \leq +\infty$. The evolution of the automaton is determined by the fixed rule F of the form

$$a_i^{t+1} = F(a_{i-r}^{t+1}, a_{i-r+1}^{t+1}, \ldots, a_{i-1}^{t+1}, a_i^t, \ldots, a_{i+r}^t) \tag{2}$$

with

$$F(0,0, \ldots, 0) = 0$$

The next state is thus computed using the newly updated values $a_{i-r}^{t+1}, a_{i-r+1}^{t+1}, \ldots, a_{i-1}^{t+1}$, instead of $a_{i-r}^t, a_{i-r+1}^t, \ldots, a_{i-1}^t$, as in the usual cellular automaton. This is analogous to the operation of an *infinite impulse response* digital filter, whereas a cellular automaton corresponds to a *finite impulse response* digital filter (see [4], for example).

Furthermore, the next-state rule will be fixed for each neighborhood radius r as follows. Let the total number of 1's in the argument of F be denoted by S:

$$S = \sum_{j=-r}^{-1} a_{i+j}^{t+1} + \sum_{j=0}^{r} a_{i+j}^t \tag{3}$$

Then the new value of the site is given by

$$a_i^{t+1} = \begin{cases} 1 & S \text{ even but not } 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

We will call this rule the *parity rule*, and the class of FA's with this rule the *parity-rule FA's*.

Although we allow the sites in an FA to extend from $-\infty$ to $+\infty$, we must assume that to the left, anyway, there are only a finite number of sites containing

non-zero values. This will then give us an unambiguous way to compute the evolution of the FA, using a left-to-right scan. We will always start with an initial configuration that has only a finite number of non-zero site values.

In the usual CA all sites are thought of as being updated simultaneously. This cannot be done in the implementation of an FA, because of the dependence of new site values on the new site values to the left but in the same generation. An FA can, however, be implemented in a highly parallel way by updating the site values along a diagonal frontier that extends from the the upper right to the lower left. This is equivalent to what happens when the stream of sites is processed by a cascade connection of identical processors, as described in [2].

3. Particles

A remarkable property of the parity-rule FA's is the profusion of particles they support. Before we discuss methods for cataloging these particles, we need some definitions.

Definition Let r be fixed. A *particle sequence* is a finite sequence A_k of 0's and 1's, starting and ending with 1, with the following properties: If the parity-rule FA with state a_i^t evolves from the initial condition determined by A_k surrounded everywhere else by 0's, then

$$a_{i-d}^{t+p} = a_i^t$$

where p is the smallest integer with this property, called the *period* of the particle sequence. The integer d is called the *displacement*, and the ratio d/p the *speed* of the particle sequence. Every particle sequence can be interpreted as an odd binary number in the natural way, with the most significant bit on the left. The resulting integer is called a *particle code*. The evolution of a particle sequence thus determines a p-length sequence of particle codes, which we call the *orbit* of the particle sequence. The smallest integer in the orbit we call the *canonical code* of the particle sequence, and we identify the canonical code with the particle itself. □

Figure 1 shows the evolution of four typical particles for the $r = 5$ parity-rule automaton.

4. Cataloging Particles

In this section we will describe two methods for finding particles: the first uses a simple enumeration strategy, and the second a constructive algorithm.

In the enumerative method, the integers from 1 to some fixed large number, say $N_{\max}$, are decoded into binary bit patterns and used to initialize the sites in an FA for a given rule-radius r. The FA is then run forward a fixed number of generations, and the resulting bit patterns analyzed for periodic sequences. Those periodic sequences are then parsed into sequences that have no internal gap of consecutive 0's longer than $2r-1$, our criterion for classifying a periodic structure as one and not many particles. Clearly, this method systematically enumerates all possible particles: It can detect any particle if run for a value of $N_{\max}$ as large or larger than the

particle's canonical code. The enumerative method lends itself to the dictionary approach, where all particles up to some length are produced at one time and stored for later use. That is how the main data base of particles used later on was produced.

Particle histograms based on these dictionaries are given in [1] for widths up to 16 bits, and radii from 2 to 6. The number of particles with canonical code less than a given width increases sharply with the value of r. For example, for $r = 2, 3, 4, 5, 6$ there are 8, 198, 682, 6534, and 13209 distinct particles with canonical code-widths up to 16 bits, respectively.

The enumerative method has a few drawbacks, all springing from the fact that the initial bit patterns are not strongly related to the particles that they generate. First, the time required to classify the particles that result from each initial pattern is rather large, although in practice it is almost constant as a function of particle length. Second, to generate all particles of length up to L requires $N_{\max} = 2^L$ iterations of the main loop, so large particles cannot be generated in practice. Since the mapping of initial seeds onto particles is many-to-one, it is reasonable to hope for a more efficient method. Last, the method does not lend itself to modifications for finding specialized types of particles more efficiently. That is, if we wanted to search only for particles with some special characteristic, such as speed or period, we would still be required to find all the other particles with the same width as well.

The remainder of this section is devoted to describing the constructive algorithm, which can generate very large particles, and which is also not without some theoretical interest. The algorithm generates all FA particles for a given radius r whose periods divide a given number. Among other things it will enable us to generate all particles with period 1, those which have only one orbital state. The running time is only linear in the maximum size of the particle being searched for, as opposed to exponential for the enumerative method, and this allows particles of length over 100 to be discovered quickly, as opposed to the practical limit of around 18 imposed by the first method.

As a prelude to the statement of the algorithm, consider a particle as a two-dimensional array of bits, with time as the vertical axis. Now consider the "leftmost edge" of a particle to be the pattern formed by the leftmost bit of each orbital phase of the particle over one complete cycle. The algorithm is based on the observation that, given the leftmost edge of a particle, the parity update rule uniquely determines the entire particle.

Actually, the truth of this observation involves a refinement of our idea of "particle". Recall that for the first classification method, a particle is considered to be anything periodic without a gap of $2r$ 0's in it. This leads to some discrepancies, like multiple particles traveling with a gap of $2r$–1 0's between them being considered one particle when they obviously should not be. The definition suggested by the forthcoming algorithm eliminates such particle-groups from consideration as particles, and also draws attention to an interesting phenomenon, known as "resonance". When two particles of the same speed, at the proper relative orbit phases, are placed just far enough apart so that an interaction does not occur in the first generation, but does occur in a later generation due to variations in the relative spacing of the particles at different orbital phases, they sometimes (very rarely) resonate, interacting only

to return to their starting positions and interact yet again. Such structures were classed as individual particles by the enumerative method, which led to their their discovery as supposed "particles" which the new algorithm did not generate.

We now describe the algorithm. For our two-dimensional pictures, let time increase downward. Now consider a given FA "update window" over a section of the particle. It looks like:

```
                        z  1  ...  r-1  r
-r  -r+1  ...  -1       z
```

where z represents the "center" of the window. Now in an ordinary update procedure the contents of all but the bottom z position are known at each step, and this is filled in according to the number of 1's in the remaining cells. However, suppose that all cells are known except the r cell at the upper right. Then its contents are obviously determined by the remaining cells in all but the two cases where everything in the window except perhaps the lower z cell is filled with a 0:

a. If the lower z contains a 1, then there is no way to fill in the upper r cell and be consistent with the update rule. When this situation is reached in the algorithm, it means that no particle with the current left edge exists, so another is tried.

b. If the lower z contains a 0, then the upper r cell can be filled with either a 0 or a 1 and still be consistent with the rule. In the algorithm it is filled with a 0. This corresponds to weeding out multiple particles traveling in parallel, and resonating particles.

Now it becomes clear how the algorithm proceeds. There are only r^p possible left ends to a period-p particle, because of the "speed of light", p–1. Furthermore, such initial configurations can be generated easily in lexicographic order, complete with r 0's to the left of each initial 1. We begin, then, with p–1 (staggered) stacked "windows", the top of each one residing on its own orbital row (excluding the bottom row), each having one of the initial 1's in its upper r–1 positions. Then, as described above, we proceed upward from the bottom window, filling in the second cell of the particle at each orbital position. When we have filled in the second cell of the top orbital position, we copy the contents to the second cell of the bottom orbital position, to enforce the period-p condition. Then we advance all of the windows one step, and proceed for succeeding iterations just as in the first. This main loop terminates when one of the following three conditions is satisfied:

a. A gap of length $2r+3$ is found in the particle, in which case it is recorded and the next initial configuration is considered.

b. A state is reached where there is no possible way to fill in one of the cells, in which case the next initial configuration is considered.

c. The maximum particle length being searched is exceeded, in which case the next initial configuration is considered.

Thus an outline of the algorithm is:

```
for each of the r^p left edges
    while ( a. max particle length not exceeded and
            b. dead end not reached and
            c. particle not isolated )
      begin
        update windows;
        record particle if appropriate
      end
```

The complexity of the algorithm is easily seen from the above outline. There are r^p steps, each consisting of a maximum of L (= maximum desired particle length) iterations of a constant time loop. Therefore the "practical" complexity of the algorithm is best stated as $O(r^pL)$, especially since we are usually concerned with particles smaller than some given value of L; for instance $L = 32$ gives all particles which are codable by an integer on a VAX, or $L = 80$ gives all particles whose pictures fit on a CRT screen.

However, we can use the algorithm to generate all particles of period p regardless of size. To see this, consider the state of the construction procedure at a given instant. Note that the current state, and thus the entire future of the procedure, is completely determined by the contents of the current windows. Thus an exact repetition of the contents of all of the window cells at some time during the procedure is equivalent to non-termination of the main loop (if the maximum length condition is dropped), since then the same repetition would occur forever. This is clearly equivalent to the non-existence of a period-p particle with the left end which is currently being considered. Now there are $r \times p$ window cells, so since each can have only two possible values, there are 2^{rp} distinct contents. Thus there must be some repetition of contents some time within the first 2^{rp} iterations of the main loop, assuming that no particle was found before then. At that time we may conclude that no particle is forthcoming, and go on to the next initial left end. Thus taking $L = 2^{rp}$, we have an algorithm for generating all particles of period p for a given r, with the time complexity of $O(r^p 2^{rp})$.

As a bonus, the argument above establishes the following fact.

Theorem 1 The number of particles with period p for any radius-r filter automaton is no greater than r^p. □

5. Coding Information in Particle Phase

A particle can be viewed as carrying information in the following way. Assume that we establish an absolute origin in space ($x = 0$) and time ($t = 0$), and a reference particle that starts in the state corresponding to its canonical code with its left end at $x = 0$. Given any instance of a particle, run it backwards in time to $t = 0$ and measure its orbital state s and the position x of its left end. The orbital state so obtained will be called the *orbital phase* of the particle, and will take on values $s = 0, 1, \ldots, p-1$, where p is the period, and $s = 0$ corresponds to the canonical code of the particle. The position x so obtained is called the *translational phase*. Thus, when no collision takes place, the orbital and translational phases of a particle both

remain 0. We might wish to keep track of the translational phase only modulo the particle's displacement d, in which case the particle has $p \times d$ distinct phase states.

We will choose to do things a little differently, however, because we will be studying the effects of collisions between pairs of particles. To go further we need to discuss collisions in some detail. First, we note that if two particles start close enough together, it may happen that they interact in a way that is impossible when they start far apart. In such cases we say the collision is *improper*; otherwise we say it is *proper*. We will restrict our attention to proper collisions, because we will always provide an initial spacing large enough to ensure a typical collision. Next, two collisions will be said to be the *same* if they can be put in concordance by a shift in space and time. Otherwise, they will be *different*. Finally, we need the following fact, which is proved in [1].

Theorem 2 Let two particles have displacements d_1, d_2, periods p_1, p_2, and speeds $d_1/p_1 < d_2/p_2$, so that particle 2 can hit 1. Let the difference in speeds be $\Delta s = d_2/p_2 - d_1/p_1$. Then the number of different proper collisions is no larger than

$$DET = p_1 p_2 \cdot \Delta s = p_1 d_2 - p_2 d_1$$

We call DET the *determinant* of the collision, it being the 2×2 determinant with rows p_1 p_2 and d_1 d_2. □

It is now clear that if we have particles with only two different displacement-period pairs, then it is the relative positions of particles modulo DET that count, insofar as the results of collisions are concerned. For this reason we will measure translational phase modulo DET, and a particle in such a collision system will have $p \times DET$ phase states.

In what follows we will place particles in the initial array of the automaton at predetermined positions and in predetermined orbital states. Collisions will then take place, and the final results of those collisions will arrive at some arbitrary fixed position far to the left, which will be thought of as the "output" position. We assume that an observer can measure the time of arrival and orbital state of each particle that arrives at the output position, and thereby determine its orbital and translational phase.

6. Collision Tables

We now describe a table that determines the results of pairwise collisions between two given particles. The particular form of the table is not the only one possible, but is one that will be useful to us in the carry-ripple example. The table is constructed empirically, by simply simulating the parity-rule automaton.

A typical table is given below for collisions in the $r = 5$ parity-rule filter automaton between the code 155 particle with displacement-period pair (22/8), and the faster code-11 particle with $d/p =$ (9/3). There are 3 sections in the table, each corresponding to a possible initial orbital state of the fast particle; the codes in the orbit of the fast particle are given in the headings, namely 11, 25, 37. Within each

section there is 1 row for *DET* consecutive spacings, starting with a value large enough to ensure that all collisions are proper; in this case $3 \cdot r = 15$. The numerical value of the spacing is the number of cells between the right end of the slow particle and the left end of the fast. In each row we record the code of the resulting fast particle, and if it is in the orbit of the original fast particle (that is, if the identity of the fast particle is unchanged by the collision) we follow this with the changes in its orbital and translational phases ($\Delta\phi_o$ and $\Delta\phi_t$). The same information for the slow particle follows in that row.

start 155 22 8				11 9 3		
spacing	code	$\Delta\phi_o$	$\Delta\phi_t$	code	$\Delta\phi_o$	$\Delta\phi_t$
15	11	0	2	167	-	-
16	11	2	3	155	2	2
17	11	1	2	155	0	0
18	11	2	3	155	2	2
19	11	2	3	155	2	2
20	659	-	-	3	-	-
next 155 25						
15	11	2	4	155	2	2
16	659	-	-	3	-	-
17	11	0	2	167	-	-
18	11	2	4	155	2	2
19	11	1	1	155	0	0
20	11	2	4	155	2	2
next 155 37						
15	11	2	5	155	2	2
16	11	2	5	155	2	2
17	659	-	-	3	-	-
18	11	0	2	167	-	-
19	11	2	5	155	2	2
20	11	1	3	155	0	0
end						

Table 1. An example of a state-transition table for the collision of two particles. Each of the three sections corresponds to a different initial orbital state of the fast particle (namely 11, 25, and 37); each row within each section corresponds to an initial spacing; $\Delta\phi_o$ corresponds to the shift in orbital phase; $\Delta\phi_t$ corresponds to the shift in translational phase, measured positive to the left for the slow particle, and positive to the right for the fast. When the identity of a particle changes, the phase shifts are undefined.

Because there can be only *DET* distinct collisions, it should be clear that the second and third sections of the table are derivable from the first, given the displacement-period pairs of the particles and the widths of the orbital codes. We will see that this redundant form of the table is convenient when searching for useful logical operations, as described in the next section.

This table has all the information we need to predict the result of any collision between a fast and slow particle, given that in the initial state the slow particle is in its canonical state. As an example, suppose that the fast particle (11) is in orbital state 25 and its left end is positioned 101 cells to the right of the right end of the slow particle, which is in its canonical state of 155. The progression of states is periodic modulo the determinant, 6, so we need to enter the table at the row corresponding to a spacing of $17 = 101 \ mod \ 6$. The entry in the code-25 section of the table then shows that the fast particle emerges with its identity intact, and with an orbital phase shift of 0 and a translational phase shift of 2, while the slow particle is changed to one with canonical code 167.

7. Searching for Logical Operations

There are many ways we might try to encode and process information using solitons, and we will describe here only one, our goal being to demonstrate the principle.

In the basic scheme we will study, the fast particle is in one of two states, identified with "0" and "1", and it passes through a number of composite structures, each composed of one or more slow particles in their canonical states but with variable spacing, and with identical displacement-period pairs. (See Fig. 2.) These composite structures will be called *particle bundles.*

Because the slow particles are all in their canonical states, the results of all the collisions can be predicted from the collision tables described in the previous section, and in fact the "0" and "1" states of the fast particle can be associated with two particular rows of the table. The problem is to choose the two rows so that the effects of collisions correspond to the logical operations we wish to perform.

We can automate the search for the slow particle bundle as follows. Create a directed graph $G = (V, A)$ with one node for each pair of rows in the collision table. Choose a fast particle and a set of slow particles with a given displacement-period pair. Suppose passage of the fast particle through a particular slow particle, at a particular spacing offset relative to an arbitrary reference (see Fig. 2), maps row i to row i' and row j to row j'. Then create an arc that goes from the node corresponding to row-pair $i\,j$ to the node corresponding to $i'\ j'$, and label this arc with the code of the slow particle and the spacing offset that produced this result. Do this for all possible collisions between the given fast particle and the given set of slow particles.

If now we want to map the fast particle state "0" to "X" and "1" to "Y", where X and Y are each 0 or 1, we need to find a path in the graph G that goes from a node of the form AB to a node of the form CD, where C = A if X = 0, C = B if X = 1, and similarly for D and Y. (For example, if we want the complementation operation we search for a path from a node of the form AB to a node of the form BA.) Each arc on the path represents a slow particle at a particular spacing offset in the bundle that we need to pass through to effect the operation.

The problem of implementing a particular set of logical operations thus reduces to that of finding a corresponding set of paths in a directed graph. The graph has $(DET \cdot p_2)^2$ nodes, where p_2 is the period of the fast particle; and DET arcs leaving each node for each slow particle allowed in a bundle. We should use breadth-first search to implement the path-search so as to find paths with the minimum number of arcs.

To make these ideas more concrete, we describe the embedding of a simple carry-ripple adder.

8. The Carry-Ripple Adder

To implement a carry-ripple adder, we will encode each pair of addend bits to one of three different particle bundles: w for the case when the addend bits are both 0, x when one is 0 and the other 1, and y when they are both 1. (See Fig. 3. for an example.) The resulting sequence of particle bundles is then transmitted (at the slow speed of the particles used to construct the particle bundles) to the left, most significant bit-pairs first. They are separated by a distance that is 0 *mod DET*, so that the collisions with the fast particle will be in accordance with the collision table; but a distance large enough to ensure proper collisions. The fast particle, which represents the carry bit, is then sent in from the left, at an initial spacing that is equal *mod DET* to the initial spacing chosen for the collision table (in the example of Table 1, a spacing of 15).

Given this particular structure, It is now easy to write down the transition rules that the collisions between the fast particle and the slow-particle bundles need to satisfy. Letting c be the state of the carry bit, we require

$$wc \rightarrow 0w'$$
$$xc \rightarrow cx'$$
$$yc \rightarrow 1y'$$

In these rules, wc represents the configuration with the particle bundle w to the left of the carry bit c, the arrow indicates the result of the collision, and $0w$ means that the fast particle is moved to its "0" state, and emerges to the left of the particle bundle w' .

For this scheme to operate properly as an adder, we must be able to decode the resulting slow-particle bundles, and that means that the result w' when $c =$ "0" must be different from w' when $c =$ "1", and similarly for x' and y' . We call this the *distinguishability* criterion. Finally we require that all the particles taking part in collisions have their speed unchanged by the collisions, so that the results of the collisions can be easily interpreted.

The graph-search problem that results from this formulation of the carry-ripple adder is not difficult, and many solutions have been obtained. One such solution for $r = 5$, and two slow particles in each slow-particle bundle, is the following:

fast particle: code 11, $d/p = 9/3$
fast particle state "0" = orbital state 0, offset 0
fast particle state "1" = orbital state 0, offset 2
slow particle $d/p = 22/8$, $DET = 6$
slow-particle bundle w: code 155 (offset 2) code 155 (offset 4)

slow-particle bundle x: code 165 (offset 2) code 207 (offset 3)
slow-particle bundle y: code 155 (offset 0) code 165 (offset 2)

Figure 4 shows a typical collision in this adder, the collision yc when c is in state "0".

Another solution was found for $r = 3$, with three slow particles in each slow-particle bundle

fast particle: code 7, $d/p = 5/3$
fast particle state "0" = orbital state 0, offset 0
fast particle state "1" = orbital state 0, offset 1
slow particle $d/p = 14/10$, $DET = 8$
slow-particle bundle w: code 919 (offset 3) code 919 (offset 0) code 919 (offset 5)
slow-particle bundle x: code 919 (offset 3) code 919 (offset 0) code 731 (offset 2)
slow-particle bundle y: code 731 (offset 7) code 919 (offset 0) code 731 (offset 2)

9. Discussion

We have demonstrated that some kind of meaningful computation can be embedded in a very simple, microscopically homogeneous, one-dimensional automaton. This embedding differs in one important aspect from the ones used to show the universality of two-dimensional cellular automata. In the latter, many cells must be put in their proper initial states before computation can proceed, and bits are stored by the presence or absence of particles [5,6]. In the one-dimensional structure discussed here, all the information necessary for computation to take place can enter in bit-serial form, with all the initial states zero. The question of whether filter automata are universal, however, remains open.

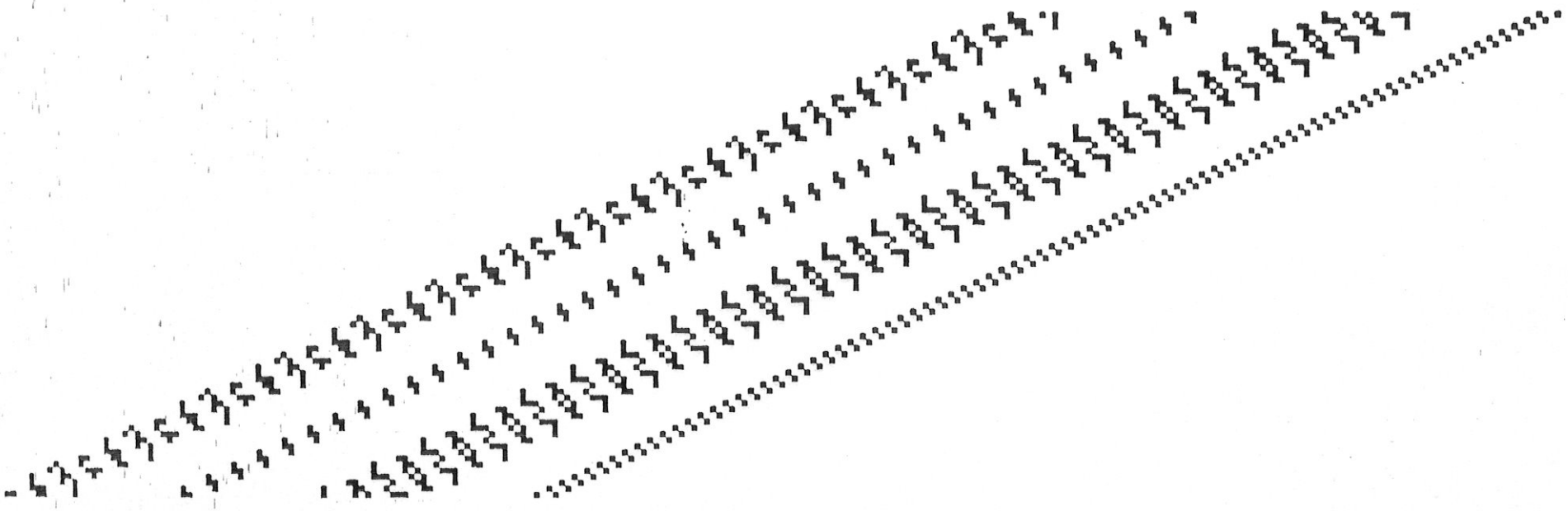
References

1. J. K. Park, K. Steiglitz, W. P. Thurston, "Soliton-Like Behavior in Automata," Tech. Rept. No. 340, Electrical Engineering and Computer Science Department, Princeton University, Princeton, NJ, April 1985.
2. K. Steiglitz, R. R. Morita, "A Multi-Processor Cellular Automaton Chip," *Proc. 1985 IEEE Int. Conf. on Acoustics, Speech, and Signal processing*, Tampa, Florida, March 1985.
3. F. L. Carter, "The Molecular Device Computer: Point of Departure for Large Scale Cellular Automata," pp. 175-194 in *Cellular Automata*, D. Farmer, T, Toffoli, S. Wolfram (eds.), North-Holland Physics Publishing, Amsterdam, 1984.
4. A. V. Oppenheim, R. W. Schafer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, N. J., 1975.
5. E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for your Mathematical Plays, Vol. 2: Games in Particular*, (Chapter 25, "What is Life?"), Academic Press, New York, N. Y., 1982.

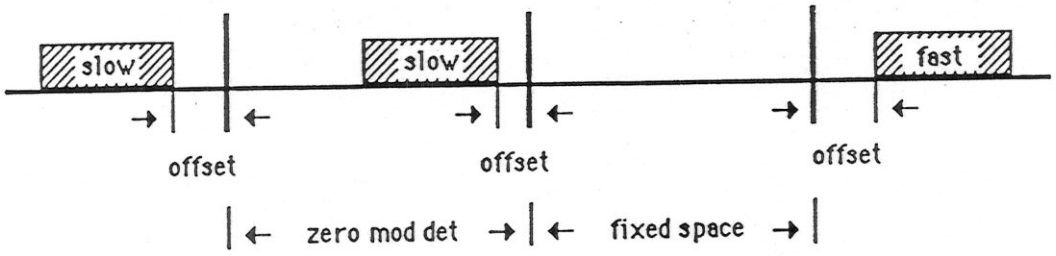
6. F. Nourai, R. S. Kashef, "A Universal Four-State Cellular Computer," *EEE Trans. on Computers*, vol. C-24, no. 8, pp. 766-776, August 1975.

Figure Captions

1. Four typical particles supported by the $r = 5$ parity-rule filter automaton, all with speed 2. From left to right the canonical codes and displacement-period pairs are 133 (12/6), 3 (4/2), 8519 (8/4), 9 (2/1)
2. The collision scheme studied here. A single fast particle passes through several slow particles in their canonical states, comprising a particle bundle.
3. An example of the form of a carry-ripple adder. The fast particle travels through the slow-particle bundles, propagating the carry bit.
4. The collision implementing $yc \rightarrow 1y'$ when c is "0" — the carry bit goes high. The particle codes are, from left to right, 155 (22/8), 165 (22/8), 11 (9/3), and the incremental spacings 0,2,0. Note that the identity of the 155 is changed, to code 167 (22/8). (The picture is circularly wrapped to fit on the page.)



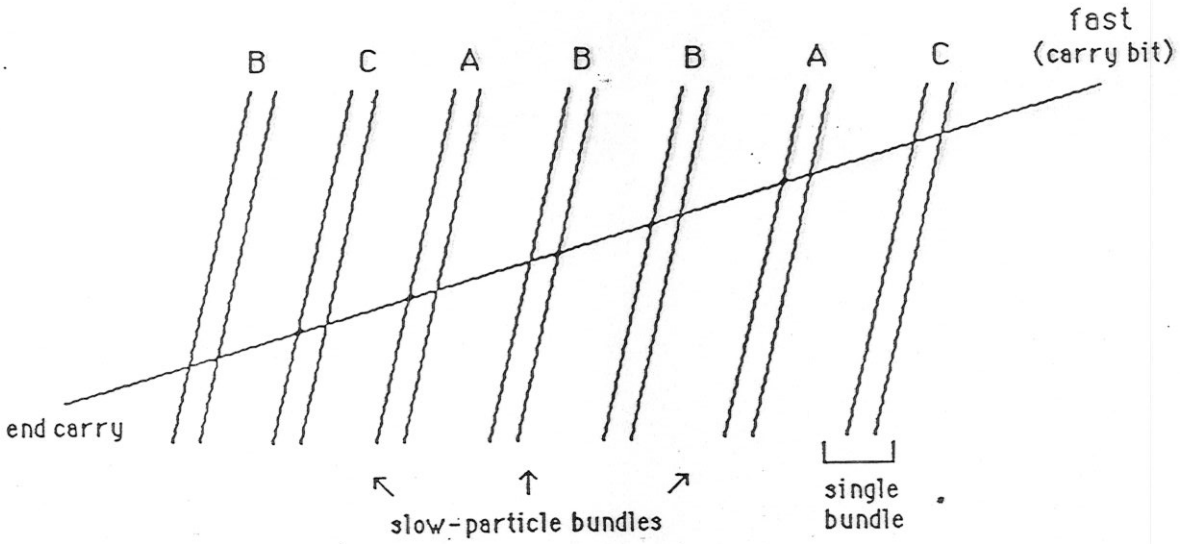
1. Four typical particles supported by the $r = 5$ parity-rule filter automaton, all with speed 2. From left to right the canonical codes and displacement-period pairs are 133 (12/6), 3 (4/2), 8519 (8/4), 9 (2/1)



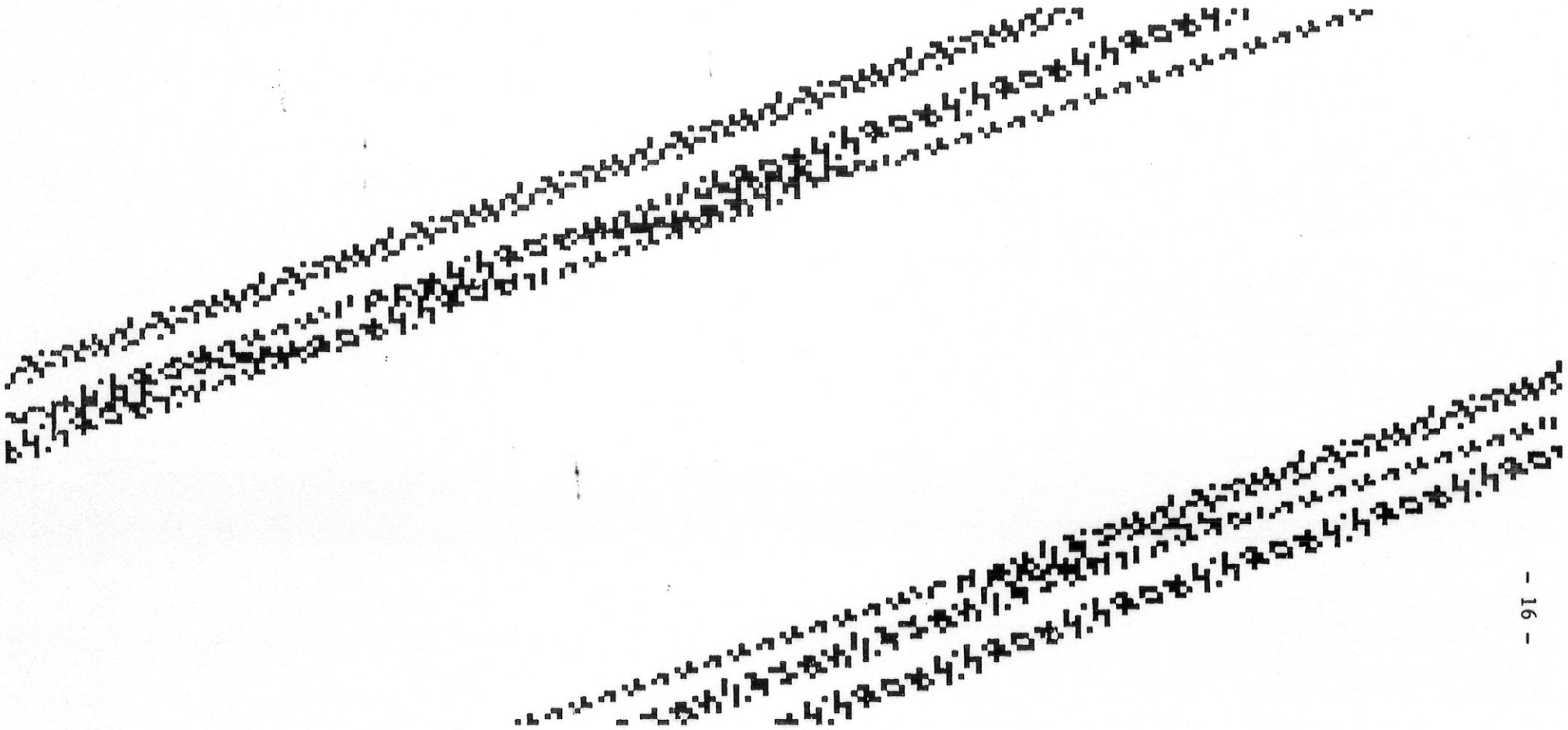
2. The collision scheme studied here. A single fast particle passes through several slow particles in their canonical states, comprising a particle bundle.

addend codes

	B	C	A	B	B	A	C	
	0	1	0	1	0	0	1	addend 1
end carry	1	1	0	0	1	0	1	addend 2
1	0	0	0	1	1	1	0	sum



3. An example of the form of a carry-ripple adder. The fast particle travels through the slow-particle bundles, propagating the carry bit.



4. The collision implementing $yc \rightarrow 1y'$ when c is "0" — the carry bit goes high. The particle codes are, from left to right, 155 (22/8), 165 (22/8), 11 (9/3), and the incremental spacings 0,2,0. Note that the identity of the 155 is changed, to code 167 (22/8). (The picture is circularly wrapped to fit on the page.)