ON THE CAPABILITY OF FINITE AUTOMATA IN 2 AND 3 DIMENSIONAL SPACE

by

Manuel Blum and William J. Sakoda

Memorandum No. UCB/ERL M78/35

June 1978

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

ON THE CAPABILITY OF FINITE AUTOMATA IN 2 and 3 DIMENSIONAL SPACE

by

Manuel Blum and William J. Sakoda*

Computer Science Division
Department of Electrical Engineering and Computer Sciences
and
Electronics Research Laboratory
University of California
Berkeley CA 94720

Reprinted from: Proceedings of the 18th Annual Symposium on Foundations of Computer Science, Providence, RI, Oct. 1977; pp 147 - 161.

Introduction

This paper considers the problem of whether a finite collection of finite automata can search all of a 2- or 3-dimensional obstructed space. Such a space gets searched by having every "accessible" cell visited at some time by an automaton. Techniques for solving search problems in 2 dimensions are presented. In particular, a finite automaton with 4 pebbles can search any finite 2-dimensional maze (of the sort that appears in various game books), while one with 7 pebbles can search any infinite 2-dimensional maze. In contrast, we show that no finite collection of finite automata is capable of searching every finite 3-dimensional maze.

A variety of interesting problems arise in the study of finite automata that move about in a 2-dimensional space. In such a space, especially one having complicated barriers, finite automata can perform in an interesting sophisticated fashion. A number of different theoretical devices that operate in 2-dimensional space have already been studied: M. Paterson [Pat] has invented a class of finite automata called "worms" that move through space, leave a trail wherever they go, and by restriction on the allowable programs, never pass through their own trail. In a 2-dimensional Euclidean space, these worms can generate rich and complex patterns, even though their programs are simple. Conway's [Con] Game of Life provides another example of how a few simple rules in 2-dimensional space give rise to very complex activity and, in this case, to a simplest known basis for self-reproducing machines with Universal Turing Machine capability.

* This research was supported in part by National Science Foundation Grant MCS75-21760-A01.

An old but still very strong mathematical argument due to M. Minsky [Min] demonstrates the increased power of automata in 2 compared with 1-dimensional space. Consider a finite automaton that moves about on an infinite 2-dimensional checkerboard. The cells of the checkerboard are white except for those on the x and y axes which are black. An automaton, represented by a circle in Figure 1, is a finite state machine that moves about from cell to cell of the checkerboard, able to see only the color of the cell it occupies. It has a finite number of internal states and a finite set of instructions which cause it, depending on its state and the color (black or white) of the cell it occupies, to move N, E, S, or W one cell and change state.

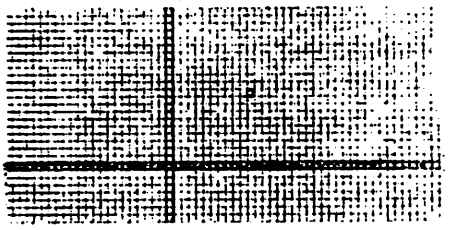


Figure 1

This finite automaton actually has the power of a Universal Turing Machine because the distance of the automaton from each of the 2 axes may be viewed as the contents of 2 counters x,y, and Minsky has shown that 2-counter machines are universal. This shows that an automaton's movements in 2-dimensional space can be considerably more complex than in 1-dimensional space, since no finite automaton is univeral on a 1-dimensional tape, no matter how that tape is marked.

In section 1, we show that finite automata can search all of 2-dimensional obstructed space. In this case, we view the automata as ants traveling about on

dry land. Water, be it finite (lakes) or infinite (oceans), constitutes the obstructions. The land too may be finite (island) or infinite (continent). This land-search problem is trivial if the automata are replaced by Turing machines: A single Turing machine can construct an internal map of its space, keep track of each cell of the space that it visits, and schedule itself to visit increasingly larger portions of (accessible) land. Of course, this solution requires memory proportional to the amount of space to be visited. Our solution by finite automata shows that finite memory distributed among a finite number of machines is sufficient. The main difficulty in constructing such an algorithm lies with the obstructions. In fact, an unobstructed everywhere infinite 2-dimensional checkerboard can easily be searched by 2 finite automata and a single pebble. Figure 2 suggests one simple algorithm.

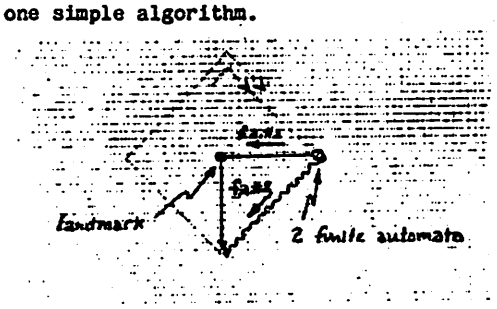


Figure 2

In fact, 2 finite automata and 1 pebble can simulate a universal 2-counter automaton (Sipser [Sip]). This yields a more powerful method for searching space along lines first suggested by A. Meyer, whereby the automata compute a search path and move along it. Cobham has shown and we have independently proved that the slightly weaker collection consisting of 1 finite automaton with 2 pebbles has not got the power to search all space: the finite automaton can use its 2 pebbles to search any sector of an infinite 2-dimensional checkerboard, if the sector's interior angle is less than 180 degrees. However, no single finite automaton with just 2 pebbles can search a complete half plane (no less the whole plane). The proof of this fact also shows that 1 finite automaton with 2 pebbles cannot be universal. The above results (concerning 2 finite automata with 1 pebble and 1 finite automaton with 2 pebbles) completely summarize the minimum finite automaton power required to search an unobstructed checkerboard.

The search algorithms for 2-dimensional space are particularly interesting in view of the difficulty of searching more general graphs. In his groundbreaking work of 1967, M. Rabin [unpublished] showed that a finite automaton with a finite collection of pebbles cannot entirely search (thread) an arbitrary finite graph. S. Cook [Coo] and C. Rackhoff [Rac] have since obtained upper and lower bounds on the number of pebbles needed to search a graph with n nodes. In section 2 we show that a finite collection of finite automata cannot completely search a 3-dimensional checkerboard space containing obstructions (the arcs and nodes of Rabin's graph become the accessible region of this space, the space between them becomes the obstruction). This extension is interesting because in checkerboard space, unlike graph space, an automaton has a "compass" for determining direction, N, E, S, W, U, D, and as we shall see in the search algorithms for 2-dimensional space, this can provide surprisingly useful information.

Bob Tarjan has informed us that he and Wolfgang Paul tried unsuccessfully to prove that a finite collection of finite automata cannot entirely search a planar graph, one whose nodes are all of degree 3. (A finite automaton moving into a node of such a graph may choose to go left, right, or back whence it came, depending on its state and whether or not other automata appear at the same node.) We suspect, as Paul and Tarjan do, that a search procedure for this related problem is impossible in general. It would be interesting to show this is so, especially since it would illuminate the difference between graph space and checkerboard space.

1 Searching 2-Dimensional Space

1.1 Overview.

In this section we show that obstructed 2-dimensional checkerboard space can be completely searched by one finite automaton with a finite number of pebbles. The search procedure uses several ideas, outlined below.

First, suppose a finite automaton with 4 pebbles is positioned on the south shore of a lake [the (south) shore of a lake $=_{df}$ all land cells that are adjacent at an (northern) edge to a cell of the lake]. That automaton can find its way to the nearest accessible land, if any, that lies due north (on the other side of the lake) of the starting position, cf. Figure 3. The finite automaton does this by using shoreline distance [shoreline distance between 2 land cells on the shore of a body of water $=_{df}$ number of land cells on that shore that connect the given 2 cells] between a pebble, W, and each of 3 other pebbles, X, Y,

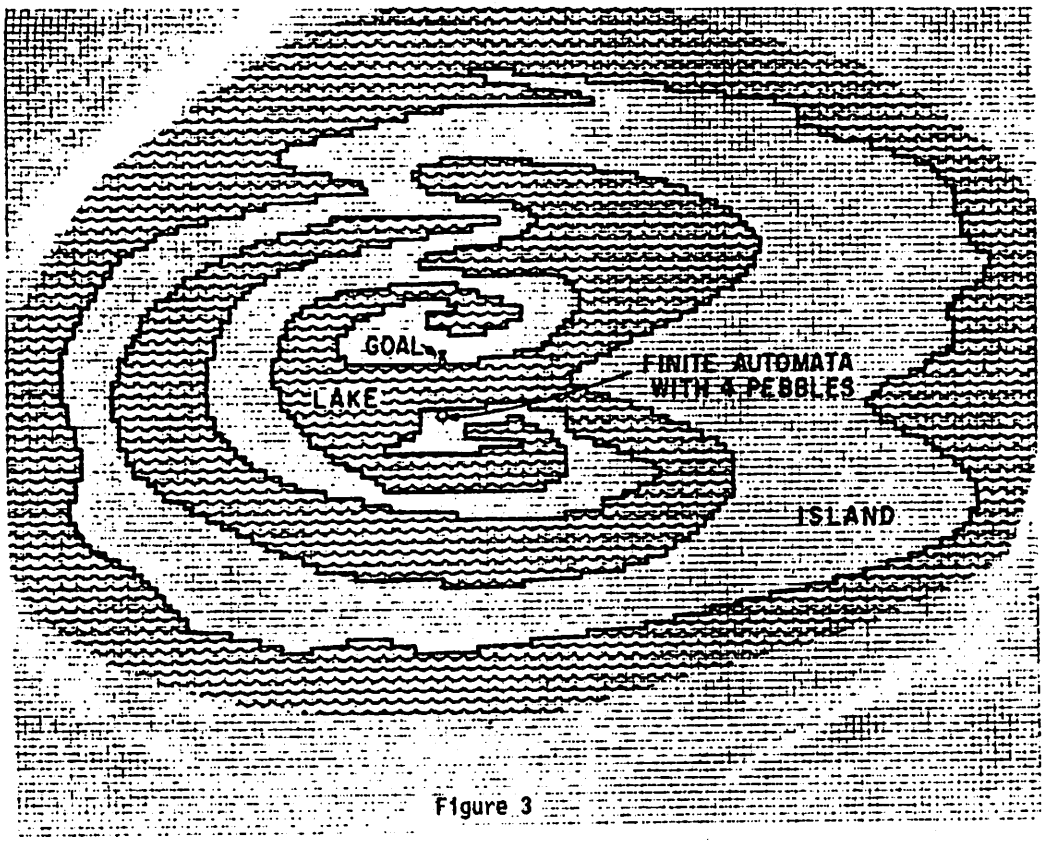


Figure 3

Z , to encode the contents of 3 finite counters, C_x, C_y, C_z. A simple proof shows that a finite automaton can cross a lake when provided with 3 such counters, each capable of holding a number no bigger than the length of the lake's shoreline. C_x, C_y are used to store x, y distance from the initial position of the automaton on the shore of the lake to successive positions of the automaton along the shore. C_z is used to (eventually) store z, the y distance to the desired goal position. The number z is the least positive y that occurs each time x = 0 as the automaton moves along the shore. (The automaton uses the empty counter, C_x, to compare the contents of C_y with those of C_z and to update C_z.)

More generally, the finite automaton may be started with its pebbles on the southern shore of an arbitrary body of water, be it finite or infinite, in what we call a "try to cross the water" state. If the shoreline is infinite, the automaton will move (with its pebbles) forever along the shore. If the shoreline is finite, the automaton will move along that shore just until it returns to its original starting position. The finite automaton will always recognize when it returns to its starting position and this will cause it to enter a predetermined "back to the starting position" state. If there is accessible land due north of the starting position, the finite automaton will recognize that fact (z, the contents of C_z, will be positive). In that case, the automaton will move along the shore to the other side of the lake (the first cell where x = 0 and y = z) then enter a predetermined "goal achieved" state.

Interestingly enough, it is just the above obstacle-crossing subroutine that fails in 3 dimensions: no fixed number of finite automata can cross the kinds of complicated obstacles that can occur there (in 3 dimensions).

The (above) obstacle-crossing subroutine can be used by a finite automaton with 4 pebbles in an algorithm to completely search any island on which it and its pebbles are placed. Basically, the automaton uses its pebbles to search column after column of the island from its original starting position to the eastern end of the island and from there to its western end.

The question is open whether 1 finite automaton without pebbles or even whether 4 automata witnout pebbles can search an arbitrary island.

Next, additional pebbles are introduced to search an arbitrary land whether it be (finite) island or (infinite) continent. These additional pebbles serve to define counters whose contents determine the size of an artificial island that is seached, enlarged, then searched again (Figure 6).

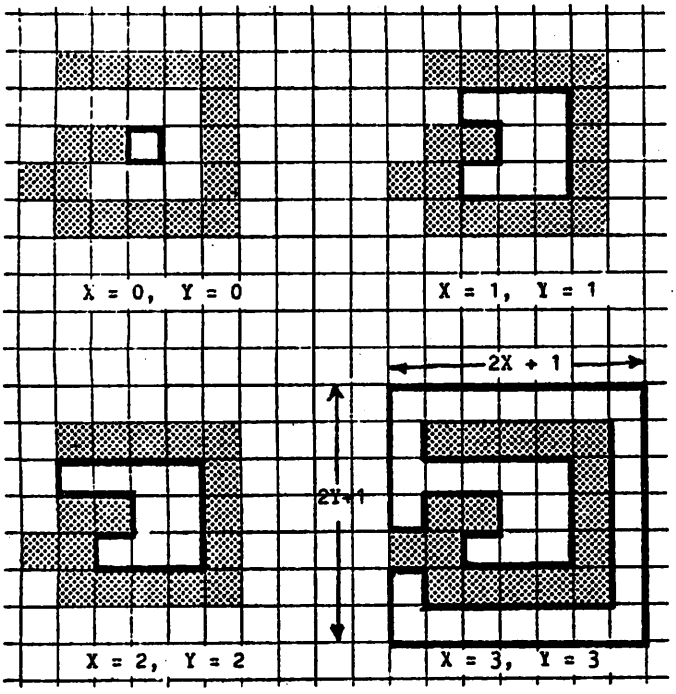


Figure 6

This process continues forever, if the accessible land is infinite. If it is finite, the finite automaton eventually realizes this and reverts back to the island searching routine.

The problem of creating an artificial island reduces to that of constructing a counter in obstructed space from a finite collection of pebbles. In such a counter, an integer is stored as the "distance" properly defined between a stationary "origin" pebble O and a "count" pebble C . A squadron of 4 pebbles under the control of 1 finite automaton moves between the 2 pebbles O and C to increment or decrement the count. A properly constructed counter is the most subtle (if one may call it that) part of this algorithm. This is because a surprisingly large collection of convincing constructions fail the test of proof. The outline for a correct construction appears in the next subsection of this paper.

Finally, a working counter must be "movable", count included, from a given cell to an adjacent one. This is easily done by introducing a second empty counter into the adjacent cell, then successively decrementing the given counter while incrementing the adjacent one. This completes the outline of the argument.

The above finite automaton with all its pebbles moves in a sufficiently straightforward manner that one can hope to visualize its movement <u>in detail</u>. The approach, however, requires a large number of pebbles. There is another approach more frugal of pebbles that dates back to Meyer's suggestion (to Rabin) for getting universal Turing machines to thread unobstructed space. That approach, though too abstract to visualize in detail (because it requires computing paths), can be used to prove that finite automata with just 7 <u>pebbles</u> can search 2-dimensional obstructed space. The approach uses the fact that a finite automaton with 2 counters is universal and can therefore generate instructions to search in turn all finite paths extending from its starting position. This works if the accessible land is infinite for then each counter can store an arbitrarily large integer. If the land is finite, the automaton can revert back to the island-searching routine. The 2 counters use one origin pebble O , two different count pebbles C_1 , C_2 , and 4 additional pebbles W , X , Y , Z for shuttling between O and C_1 , C_2 . This collection consisting of 1 finite automaton and 7 pebbles can search any obstructed 2-dimensional space.

1.2 <u>Counter construction</u>.

In this section, we design a finite automaton that uses 6 pebbles when placed in an infinite obstructed 2-dimensional space to simulate a (potentially infinite) counter. The 6 pebbles consist of an origin pebble O , a count pebble C , and 4 additional pebbles W , X , Y , Z .

<u>Problem</u>: Design a finite automaton to be started in an "increment" state together with pebbles O , C , W , X , Y , Z on a single cell of land. Call this initial configuration the beginning of stage 1. In general, at the beginning or end of a stage, the finite automaton is to occupy this starting land cell with O , W , X , Y , Z , while C may lie elsewhere (storing the count). At the beginning of stage n, the finite automaton may be started in an "increment" state or else, provided C does not occupy the same land cell as O , in a "decrement" state. In either case, the finite automaton uses its pebbles W , X , Y , Z to find C , to move it, and to return to O either in a "mission accomplished" state or a "mission impossible" state. The return to O constitutes the end of stage n. If the finite automaton returns to O in the mission impossible state, this is to mean the accessible land is necessarily finite (island). In this case, the finite automaton is not to be restarted. If the finite automaton returns to O in a mission accomplished state, it may be restarted in an increment or decrement state, and this consti-

tutes the beginning of stage n+1. At the end of a stage, C is to occupy the same position as O if and only if the finite automaton has been started as often in the increment as the decrement state.

The finite automaton with its 4 pebbles W , X , Y , Z is called the "shuttle" since it generally shuttles from O to C , moves C appropriately, and then shuttles back to O . Pebbles W , X , Y , Z are used by the finite automaton as previously explained to simulate 3 finite counters C_x, C_y, C_z. All shuttle movements will be described in terms of these counters rather than the pebbles that implement them. Since these counters are used only to store values of shoreside distance, the replacement of the pebbles by these counters is legitimate. (Our description of the shuttle movements is thereby simplified because the finite automaton can update C_x, C_y, C_z contents on the spot and the corresponding movements to the various pebbles need not be described.)

We now give instructions for the shuttle (i.e. the finite automaton with counters C_x, C_y, C_z) to move from O to C .

ALGORITHM(indented):

The shuttle is to follow the instructions below until C or O is reached: Initially, the shuttle is to move due north until it reaches a cell, call it X, of the south shore of a body of water. From X, the shuttle is to move counter clockwise along the shore, updating C_x, C_y, C_z as it goes. If and when it returns to X, the shuttle shall know if there is reachable land due north of X (yes if $C_z > 0$, no if $C_z = 0$). If not, the shuttle is to return to O in the "mission impossible" state. If yes, it is to move from X to the closest land cell due north of X (i.e. to the other side of X) (cf Fig. 7).

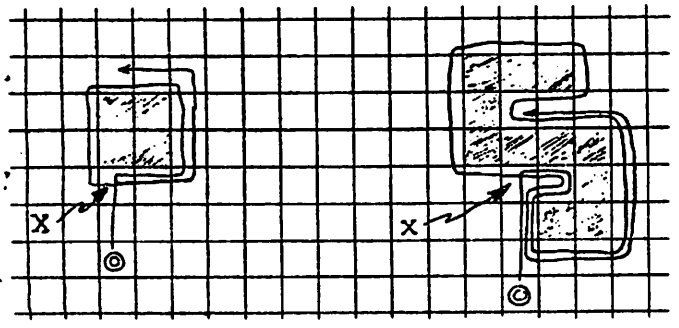


Figure 7

From this other side of the lake, the shuttle is to continue its movement north following the directions given above.

END OF ALGORITHM

The return of the shuttle from C to O follows the same path in reverse taken by the preceding movement from O to C (cf Fig. 8).

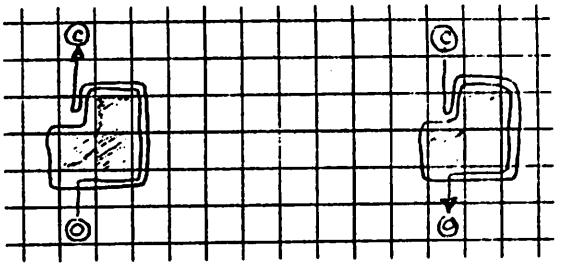


Figure 8

The C_x, C_y, C_z contents at any point in this reverse movement, however, may be different from their contents at the same point in the forward movement.

The count pebble C can be placed in more than one position on a land cell, unlike pebbles W , X , Y , Z and O . In addition to the standard position in the interior of a cell, C can also be placed on an edge between a land cell and water cell. The latter irregular position permits the shuttle to distinguish when C is on a shoreline (cf Fig 9).

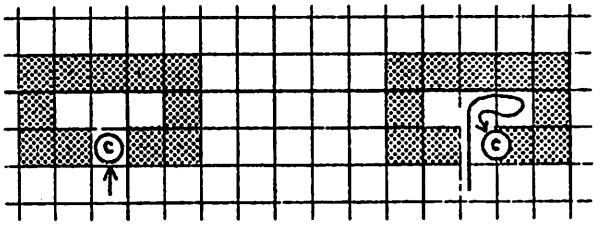


Figure 9

The position of the count pebble C relative to the origin pebble O , between stages when the shuttle has returned to O , shall uniquely determine the contents of the counter being constructed. This position will be called the between-stages position of C . Shown in figure 10 are the counter contents or numbers represented by a succession of between-stages positions.

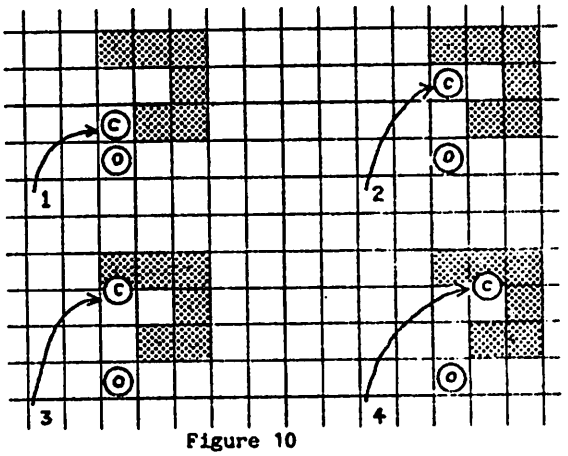
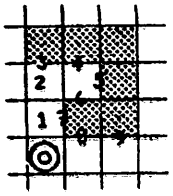


Figure 10

A single figure that supplies the above information is provided by:



Note that the successive positions of the shuttle as it moves from O to a distant C are different from the successive between-stages positions of C : C can be placed on an edge, whereas the shuttle moves only from cell interior to cell interior. The shuttle can pass through a cell repeatedly in one shuttle, whereas successive between-stages positions of C are unique. For example, the successive positions of the shuttle as it moves from O to C are compared in figure 11 below with the successive between-stages positions of C as it is incremented stage after stage.

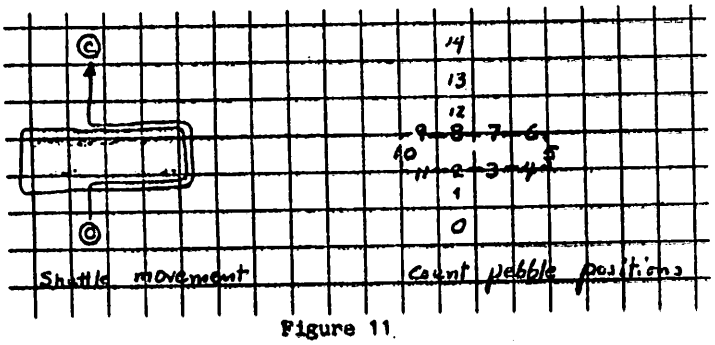


Figure 11

Note the jump in figure 11 from C 's position 11 to 12: This jump is needed to insure that no position of C will have double meaning. Another such example is given in figure 12.

As the shuttle moves from O to C, it may visit a particular cell, in particular a lakeshore cell, arbitrarily many times.

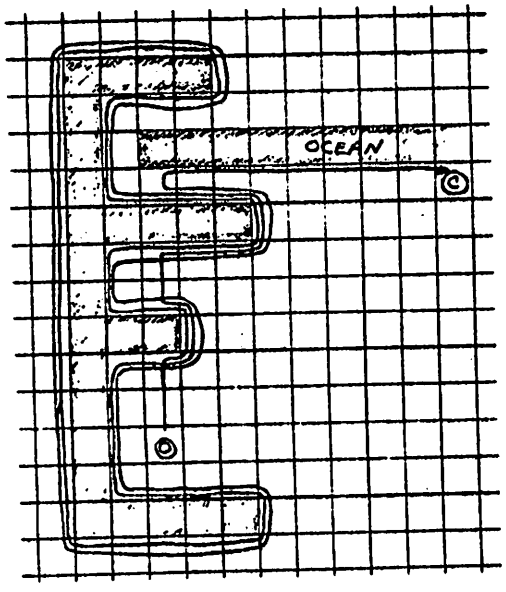
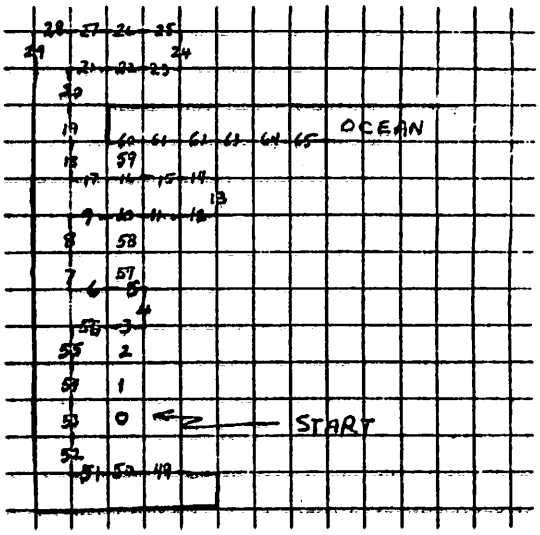


Figure 12

The sequence of C positions, however, are all different:



Notice the jumps from 56 to 57 and 58 to 59. Each integer represents the contents of the counter when the count pebble C is placed at that location.

Of course, the shuttle does not have wings. It cannot move the count pebble instantly from position 56 in figure 12 to position 57. It does this gradually instead.

The procedure used by the shuttle to <u>increment</u> the count pebble is as follows:

ALGORITHM:

From O the shuttle moves toward C (north, counterclockwise around water, etc.) until it "encounters" C . We say that <u>the shuttle encounters C</u> if and only if either [C lies inside a cell and the shuttle has just moved north or south, not following a shoreline, into the cell containing C] or [C lies on an edge (between water and land) and the shuttle has just followed the shoreside of that water into the cell on whose edge C lies].

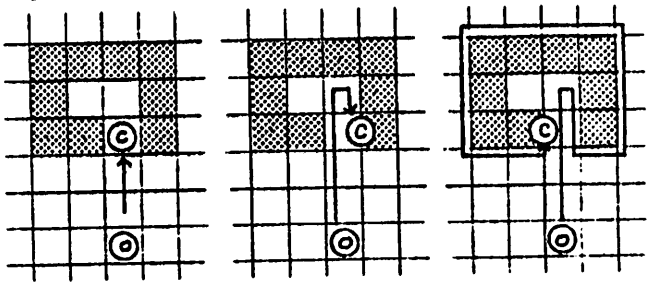


Figure 13

Figure 13 shows C inside a cell or on one of its 2 edges, and the movement of the shuttle until it encounters C .

After encountering C , the shuttle determines which of the 3 cases below holds, and moves C accordingly:

1. If C lies inside a cell (in which case it must lie in the same column as O and north of it) and if the cell north of and adjacent to the one containing C is also land, then the shuttle moves C north to the interior of the above cell (fig 14).

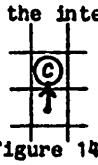


Figure 14

2. If C lies inside a cell and if the cell above is water, then the shuttle moves C up to the edge between the 2 cells (this position may get changed before this stage ends: It will be final if and only if C is not encountered by the shuttle on its way back to O).See fig 15.

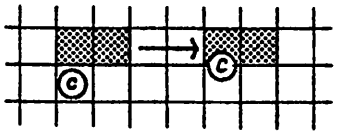


Figure 15

3. If C lies on a lakeshore edge as in figure 16, then move it counterclockwise one edge to the next position on the lakeshore. (This will be the final position for C for this stage if and only if C is not encountered by the shuttle on its way back to O.)

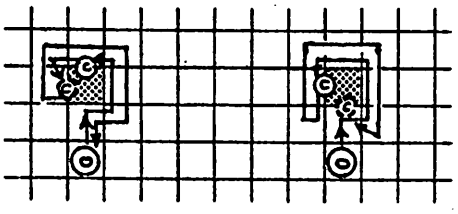


Figure 16

After moving C the shuttle wends its way back to O along exactly the same path whence it came. If C is encountered on the way back, it must be lying on the edge of a lakeshore cell (this is because the shuttle can visit only lakeshore cells more than once in moving from O to C or back). This position must be in the column containing O . The shuttle now checks if there is accessible land due north of C . If not, it returns to O in the "mission impossible" state. If yes, it moves C to the interior of the nearest land cell due north of its present position (fig 17).

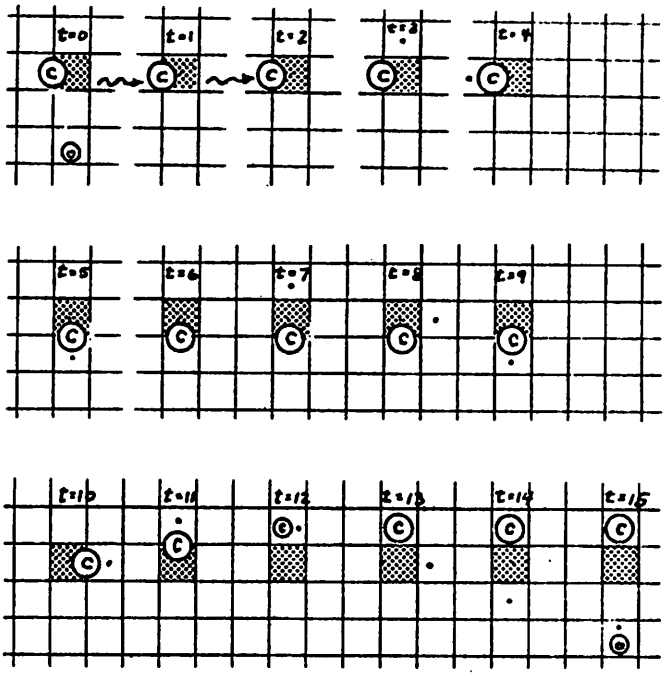


Figure 17

Now the shuttle continues its way back to O . (C will not be encountered again since it can only be encountered by the shuttle on its way back if it lies on an edge.)

END OF ALGORITHM

Decrementation will next be defined so that the counter's content (i.e. the total number of incrementations minus the number of decrementations) uniquely determines the position of C independently of the order in which the incrementations and decrementations were carried out.

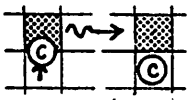
ALGORITHM(indented):

To decrement the count, O and C may not be in the same position. The shuttle leaves O and moves in the usual way north, around lakes, etc. until it encounters C . Either C is encountered in the interior of a cell or on an edge.

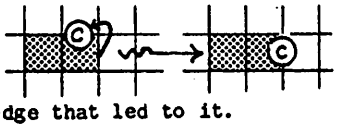
Case 1. C is in the interior of a cell and is reached by the shuttle from the interior of an adjoint cell below:

In this case, the interior of the cell below the one containing C is the desired decremented position of C .

Case 2. C is on an edge (between water and land) and is reached from the interior of the cell beneath it: Move C to the interior of that cell:



Case 3. C is on an edge and is reached from an edge:



Move C to the edge that led to it.

Case 4. C is in the interior of a cell and the southern edge of that cell is on a shore (fig 1d). In this case move C south to the edge on the other side of the water (the shoreline must be finite) and then clockwise one edge. Drop C. Then move counterclockwise one column (to the land cell in the column containing O) and start the return trip to C . If C is not encountered on the way back, then its position is final. If it is encountered, move C to the interior of the cell that lies counterclockwise in the adjacent column (column containing O) and return to O .

END OF ALGORITHM

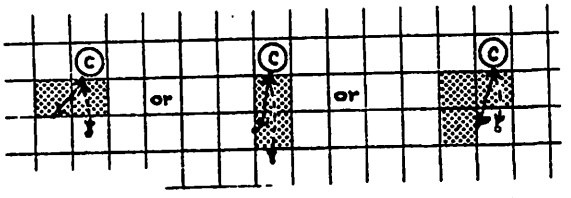


Figure 18

It is easy to see that if the land accessible from O is finite, then the shuttle will discover this before trying to move C to a nonexistent "other side" of the water. If the land accessible from O is infinite, then there is an infinite sequence of distinct positions for C , and the shuttle (properly) increments and decrements C through these positions, as we shall show in the remainder of this section.

At the start, C is in the interior of a cell, the same that contains O . This is position 0 . In general, between stages, C is either in the interior of a cell or on an edge. It is easy to see that whenever it lies in the interior of a cell, it lies in the same column as O .

1. Suppose that C is in the interior of a cell, say at position i, and that the cell above C is land. If the shuttle is required to increment C , then C will be moved to the interior of the cell above its present location. This will be position i+1 because the shuttle cannot encounter C again on its return to O . (In moving from C to O , the shuttle can encounter C a second time only if C is on water's edge.)

2. Suppose that C is in the interior of a cell, say at position i, and that the cell above it is water. If the shuttle is required to increment C , then C will be moved up to the edge between land and water. The shuttle then returns to O . If it does not encounter C on the way back, then since the shuttle is returning along the same path whence it came, it follows that this position of C is a new one. It is position i+1 . If the shuttle does encounter C on the way back then it moves C across to the other side of the water to the interior of a cell there. This position is a new one (because the shuttle does not pass through it on the way back to O). It is position i+1.

3. Suppose that C lies on an edge and that this is position i. By inductive assumption, this land cell lies on the path of the shuttle (whether placed there by incrementation or decrementation) and therefore the shuttle will encounter C . Suppose the shuttle is required to increment C . Then it moves C to the next edge in the counter clockwise motion along the shoreside and then the shuttle starts its return to O . If the shuttle does not encounter C on its return then since the shuttle is returning along the same path it came, it follows that this position of C is a new one, namely i+1. If the shuttle does encounter C on the way back then it moves C to the other side of the lake to the interior of a cell there. This position is a new one, namely position i+1.

The argument that the shuttle properly decrements C is similar to the above increment argument.

This completes the proof that a finite automaton with 6 pebbles in infinite 2-dimensional obstructed space can simulate an unbounded counter. (A finite automaton with 7 pebbles can simulate 2 counters and use them to search the space.)

2: 3-Dimensional Space is Unsearchable

In section 1, we showed that a finite state machine with 7 peboles could search any connected 2-dimensional checkerboard graph, whether finite or infinite. In 3 dimensions, the situation is different:

Theorem: Let a collection of n s-state finite state machines be given. Then there is a finite connected 3-dimensional checkerboard graph G not searchable by the n machines. If all of the machines are initially placed on the same vertex of G, there will be a vertex which is never visited by any of the machines in the ensuing computation.

Some notation for manipulating graphs and a comment about the machine model follow.

Let U = $\{(0,0,1),(0,0,-1),(0,1,0),(0,-1,0),(1,0,0),(-1,0,0)\}$.

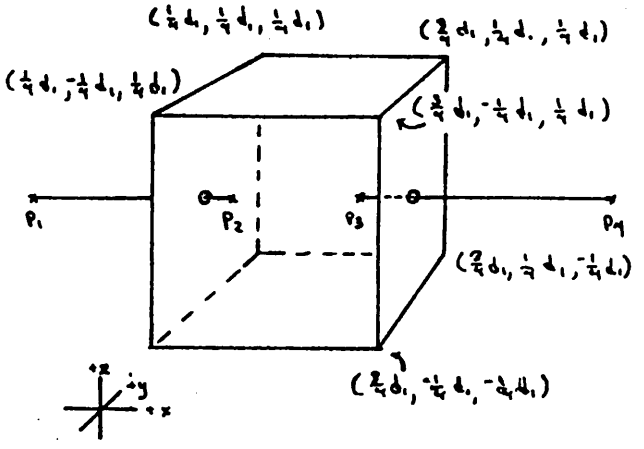
Up Down N S E W

An arc is an unordered pair $\{v,v+u\}$ where $v \in Z^3$ (Z = integers) and $u \in U$. A 3-dimensional checkerboard graph G is a set of such arcs. The set of vertices of G, V(G), is $\{v \in Z^3 \mid$ (there exists $u \in U$) $\{v,v+u\} \in G\}$. For $v \in V(G)$, the set of directions from v is $D(v) = \{u \in U \mid \{v,v+u\} \in G\}$. (This is the set of directions (N,E,S,W,U,D) along which an automaton can move from v.)

Let us describe the computation of machines $M_1,M_2,\ldots,M_n$ on a graph G. The machines are designed to cooperate together in the computation. At step t each machine M_i is located at some vertex v_i in state q_i. One or many machines may occupy a vertex at a time. Acting simultaneously, each M_i applies its transition function to its current state, the set of directions (accessible) from its current vertex v_i, and the set of machines located at v_i to obtain a direction $u_i \in D(v_i)$ and a state from its finite state set. At time t+1, M_i will be found at $v_i + u_i$, in its newly selected state.

Proof of the Theorem. The proof proceeds inductively in n stages. The i-th stage produces i-traps, which are graphs not searchable in a certain way by any i of the given machines.

Stage 1 constructs 1-traps. The plan is to generate a large collection of 1-traps (all are rotations and translations of the same basic 1-trap) (Fig. 2.1), which will later be used as the atomic units for the construction of 2-traps. We will present the construction for this 1-trap below, referring the reader to the final version of the paper for the complete proof.



$d_1 = \rho s^2 \ell^2 + 4\ell$

$\ell = \text{lcm}\{2,\ldots,5\}$

$p_1 = (0,0,0)$

$p_2 = (\frac{1}{4}d_1 + 1, 0, 0)$

$p_3 = (\frac{3}{4}d_1 - 1, 0, 0)$

$p_4 = (d_1, 0, 0)$

Figure 2.1 : Outline of a 1-trap

The 1-trap must satisfy certain physical requirements:

(i) The graph is connected.

(ii) (a) Every arc along the line between p_1 and p_2 must be present.

(ii) (b) Every arc along the line between p_3 and p_4 must be present.

The sets of vertices connected by arcs in (a) and (b) are called the wires of the trap.

(ii) (c) The wires are the only vertices which are not properly in the interior of the cube.

The 1-trap must also satisfy:

1-trapping property: Let any one of the given s-state machines be started anywhere on either of the wires. In the subsequent computation, the machine will never appear on the other wire.

A 1-trap will now be constructed. It will be convenient to begin the construction at the point (0,0,0) and add the wires later, so that the trap constructed will be a translation of the graph shown in Figure 2.1.

Definition. For x, y ∈ Z, let ⟨x,y⟩ denote the point (l*x,l*y,0) where l will henceforth denote lcm{2,...,s}. The ⟨x,y⟩ are the connector points. Let C = {⟨x,y⟩|x,y ∈ Z}.

The ∞ connector graph is constructed by joining adjacent connector points with x and y connectors.

Definition. The pair of points v_1, $v_2 \in Z^3$ lie along a line if $v_2 = v_1 + au$ for some $a \in \mathbb{N}$ and $u \in U$. For such v_1 and v_2, each arc in the set $\{\{v_1+bu, v_1+(b+1)u\} | 0 \le b < a\}$ is between v_1 and v_2.

Definition. (i) Let $c_1, c_2, \ldots, c_9$ be defined as in Figure 2.2. $C_x((0,0,0))$, the x-connector at (0,0,0), consists of all arcs which lie between a pair of points c_i, c_{i+1} for some $i \le i \le 8$.

(ii) For $p \in Z^3$,
$C_x(p) = \{\{v+p, v'+p\} | \{v,v'\} \in C_x((0,0,0))\}$. Thus $C_x(p)$ is the translation of $C_x((0,0,0))$ to p.

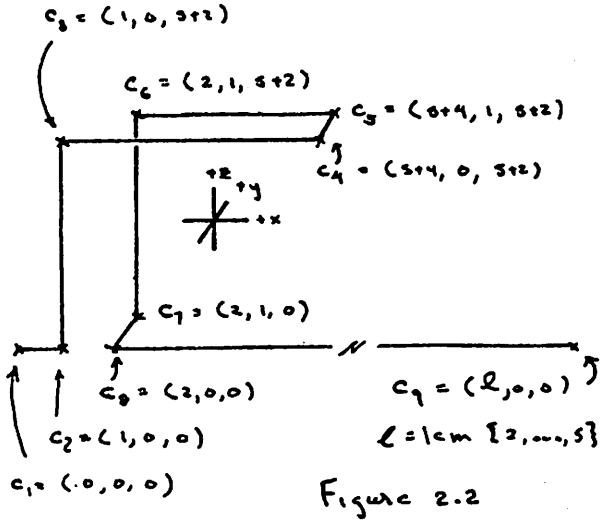


Figure 2.2

$C_x((0,0,0))$, the x connector at (0,0,0)

Definition. (i) Let $d_1, d_2, \ldots, d_9$ be defined as in Figure 2.3. $C_y((0,0,0))$, the y-connector at (0,0,0), consists of all arcs lying between a pair of points d_i, d_{i+1} for some $1 \le i \le 8$.

(ii) For $p \in Z^3$,
$C_y(p) = \{\{v+p, v'+p\} | \{v,v'\} \in C_y((0,0,0))\}$.

Definition. The ∞ connector graph is
$G_\infty = \bigcup_{p \in C} (C_x(p) \cup C_y(p))$.

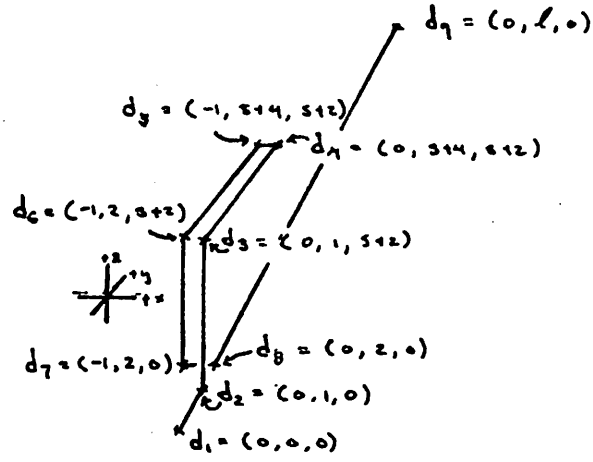


Figure 2.3

$C_y((0,0,0))$, the y-connector at (0,0,0)

The behavior of one machine on the ∞ connector graph is characterized by the ∞ Ribbon Theorem. The desired 1-trap will be obtained from a finite approximation of the ∞ connector graph. The ∞ Ribbon Theorem will be useful for analyzing the behavior of machines on the finite space.

Definition. The distance between connector points ⟨x_1,y_1⟩ and ⟨x_2,y_2⟩, denoted $d(\langle x_1,y_1\rangle, \langle x_2,y_2\rangle)$, is $|x_1-x_2| + |y_1-y_2|$.

It is straightforward to verify that d is a metric on the space of connector points.

∞ Ribbon Theorem: Let one of the given s-state machines M be started on the ∞ connector graph at ⟨x_1,y_1⟩. Then every connector point visited by M is within distance s-1 (with respect to metric d) of some point in $R(\langle x_1,y_1\rangle) = \{\langle na+x_1, nb+y_1\rangle | n \in \mathbb{N}$ and $a,b \in Z$ and $|a|+|b| \le s\}$.

The proof of the theorem uses the following property of connector space, which is easily verified.

Lemma: Let G denote the ∞ connector graph. Then for $x,y \in Z$, $G = \{\{v+\langle x,y\rangle, v'+\langle x,y\rangle\} | \{v,v'\} \in G\}$. (Thus all connector points look the same to a finite state machine.)

Proof of the Theorem. Call the i-th connector point visited by M, ⟨x_i,y_i⟩, and let q_i be the state of M at this visit. Pick the least $i < j$ such that $q_i = q_j$. Since M has only s states, $j \le s+1$. For any $k \ge 1$ the points ⟨x_k,y_k⟩ and ⟨x_{k+1},y_{k+1}⟩ are at distance at most 1, so by the triangle inequality

$\langle x_2,y_2\rangle,\ldots,\langle x_{j-1},y_{j-1}\rangle$ are at distance at most s-1 from $\langle x_1,y_1\rangle$. Set $a = x_j - x_i$, $b = y_j - y_i$. Then $|a| + |b| = d(\langle x_i,y_i\rangle,\langle x_j,y_j\rangle) \leq s$. Because all connector points look the same to the machine, the computation between steps i and j can be extrapolated, so that for any $i \leq k < j$ and $n \geq 1$:

$$x_{n(j-i)+k} = x_k + na ;$$

$$y_{n(j-i)+k} = y_k + nb .$$

Now $d(\langle x_1+na,y_1+nb\rangle,\langle x_k+na,y_k+nb\rangle) = d(\langle x_1,y_1\rangle,\langle x_k,y_k\rangle)$ which latter quantity has been shown above to be at most s-1. Q.E.D.

A finite approximation of ∞ connector space, called the marked torus, is constructed next.

Definition. $h = 4s^2$; $w = h*1$.

The construction of the marked torus uses the rectangle of connector points $\langle x,y\rangle$ such that $0 \leq x < w$ and $0 \leq y < h$. Adjacent connector points are to be joined by x and y connectors. Bridges will join pairs of connector points at opposite edges of the rectangle.

Definition. (i) Let $e_1,e_2,\ldots,e_9$ be defined as in Figure 2.4 The x bridge of span w at (0,0,0), denoted $B_x(w,(0,0,0))$, consists of all arcs lying between a pair of points e_i, e_{i+1} for some $1 \leq i \leq 8$.

(ii) For $p \in \mathbb{Z}^3$, $B_x(w,p) = \{\{v+p,v'+p\}|\{v,v'\} \in B_x(w,(0,0,0))\}$.

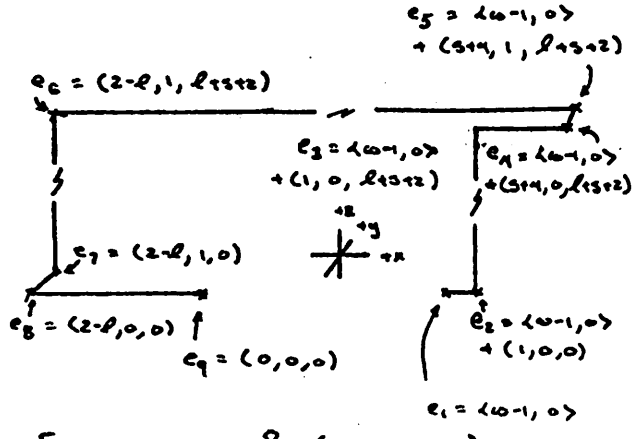


Figure 2.4: $B_x(\omega,(0,0,0))$, the x-bridge of span ω at (0,0,0)

Definition. (i) Let $f_1,\ldots,f_9$ be as in Figure 2.5. The y bridge of span h at (0,0,0), denoted $B_y(h,(0,0,0))$, consists of all arcs lying between a pair of points f_i, f_{i+1} for some $1 \leq i \leq 8$.

(ii) For $p \in \mathbb{Z}^3$, $B_y(h,p) = \{\{v+p,v'+p\}|\{v,v'\} \in B_y(h,(0,0,0))\}$.

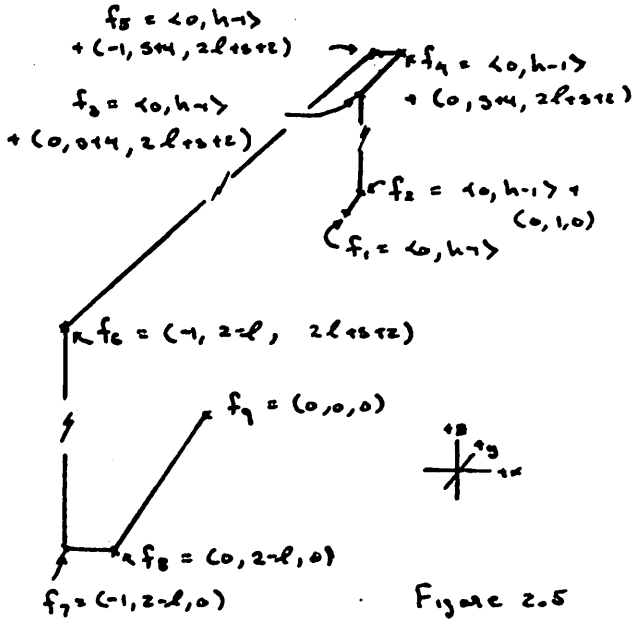


Figure 2.5

$B_y(h,(0,0,0))$, the y bridge of span h at (0,0,0)

Definition. The marked torus T is the union of the following five sets:

(i) $\bigcup_{\substack{0\leq a\leq w-2\\ 0\leq b\leq h-1}} C_x(\langle a,b\rangle)$

(ii) $\bigcup_{\substack{0\leq a\leq w-1\\ 0\leq b\leq h-2}} C_y(\langle a,b\rangle)$

(iii) $\bigcup_{0\leq b\leq h-1} B_x(w,\langle 0,b\rangle)$

(iv) $\bigcup_{0\leq a\leq w-1} B_y(h,\langle a,0\rangle)$

(v) $\{\{(0,0,0),(0,0,-1)\},\{m_2,m_2+(0,0,-1)\}\}$ where $m_2 = \langle s+1,2s^2\rangle$.

Connector points $\langle 0,0\rangle$ and m_2 are said to be marked; all other connector points are unmarked. The wires of the trap will be attached to the points (0,0,-1) and $m_2 + (0,0,-1)$.

It will now be shown that the marked torus satisfies:

Isolation Theorem: Let M, one of the given s-state machines, be started at one of the connector points $\langle 0,0\rangle$ or m_2 of T. M will never appear at the otner point during the subsequent computation.

This theorem shows that a 1-trap may be constructed from the marked torus, as follows. The torus fits properly inside a cube of side length $l^*(w+2) = 4s^2l^2 + 2l$. Lead wires into this cube as in Figure 2.1. Attach a wire to each of the points (0,0,-1) and $m_2 + (0,0,-1)$ via a sequence of arcs not passing through any vertex that is part of the torus. Let us verify that this construction will trap any one of the given machines. Suppose machine M, starting on the wire attached to (0,0,-1), arrives at the other wire. There will be a segment of the computation where M, starting at <0,0>, later arrives at m_2, having visited no marked connector points in the interim. This segment is also a valid computation of M on the marked torus (without wires), contradicting the Isolation Theorem. A similar argument shows that M cannot move from the m_2 wire to the <0,0> wire.

Here is the plan for proving the Isolation Theorem. First prove the Projection Theorem, which relates the behavior of s-state machines on ∞ connector space and the marked torus. The Projection Theorem and ∞ Ribbon Theorem togetner give the Finite Ribbon Theorem, characterizing the behavior of machines on the torus. The Marked Finite Ribbon Theorem and Ribbon Analysis provide a refined characterization, which is tnen used to obtain the Isolation Theorem.

Definition.

(i) For $m \in Z^+$ and $a \in Z$, $[a]_m$ is the least nonnegative integer such that $[a]_m - a$ is divisible by m.

(ii) For $x,y \in Z$, $f(<x,y>) = <[x]_w,[y]_h>$.

Projection Theorem: Let s-state machine M begin a computation in state q at $p_1 \in C$ on the torus. Suppose that after some number of steps of computation the sequence of connector points M has visited is, in order, $p_1,p_2,\ldots,p_r$ with $p_1,2,\ldots,p_{r-1}$ unmarked. Then M_1 started in state q at p_1 on the ∞ connector graph, will visit the sequence of connector points $p_1,p_2',p_3',\ldots,p_r'$. The relation between the two sequences is: $p_i = f(p_i')$ for $2 \le i \le r$.

Proof. Both graphs can be viewed geometrically as intersecting straight line segments. Call a point where 2 or more lines meet a corner. Associate each corner g in ∞ connector space with a corner T(g) on the torus.

(i) (a) For a, b $\in Z$ such that $a \not\equiv -1(w)$ and $1 \le i \le 8$,

$$T(<a,b>+c_i) = f(<a,b>) + c_i$$

(c_i as specified in Figure 2.2).

(i) (b) For a, b $\in Z$ such that $a \equiv -1(w)$ and $1 \le i \le 8$,

$$T(<a,b>+c_i) = f(<a,b>) + e_i$$

(cf Figure 2.4).

(ii) (a) For a, b $\in Z$ such that $b \not\equiv -1(h)$ and $1 \le i \le 8$,

$$T(<a,b>)+d_i) = f(<a,b>) + d_i$$

(cf Figure 2.3).

(ii) (b) For a, b $\in Z$ such that $b \equiv -1(h)$ and $1 \le i \le 8$,

$$T(<a,b>+d_i) = f(<a,b>) + f_i$$

(cf Figure 2.5).

In connector space M visits a sequence of m corners, being in state q_k when the k-th corner g_k is visited. On the torus, M visits a sequence of $\hat{m}$ corners, being in state $\hat{q}_k$ when the k-th corner $\hat{g}_k$ is visited.

Claim. $\hat{m} = m$, and for $1 \le k \le m$: $\hat{q}_k = q_k$, $\hat{g}_k = T(g_k)$.

The claim is verified by induction on k. The interesting cases arise when M moves between a pair of corners in ∞ connector space, both falling into cases (i)(b) or (ii)(b). For example, suppose $g_k = <-1,0> + c_7$, $g_{k+1} = <-1,0> + c_6$. By induction hypothesis $\hat{g}_k = <-1,0> + e_7$ and $\hat{q}_k = q_k$. To show: $\hat{g}_{k+1} = <-1,0> + e_6$ and $\hat{q}_{k+1} = q_{k+1}$. Let us interpret the situation in terms of Figures 2.2 and 2.4. In connector space M starts at the c_7 corner of an x-connector in state q_k and proceeds upward to the c_6 corner, arriving there in state q_{k+1}. On the torus, M begins at the e_7 corner of an x-bridge, in state q_k. To show: M reaches the e_6 corner of the x-bridge in state q_{k+1}.

For $1 \le i \le s+2$, let $v_i = g_k + (0,0,i)$. Let p_i be the state of M on the connector graph the first time v_i is visited during the passage from g_k to g_{k+1}. Since M has only s states, there exist $1 \le i < j \le s+1$ such that $p_j = p_i$. The computation then repeats according to the pattern $p_b = p_{[b-i]_{j-i}+i}$ for $j \le b \le s+2$.

For $1 \le r \le 1+s+2$ let $\hat{v}_r = \hat{g}_r + (0,0,r)$. Since $\hat{g}_k$ is unmarked, M's computation on the connector graph can be extrapolated on the torus. According to this extrapolation, M will visit each $\hat{v}_r$ on the torus. The first visit to $\hat{v}_r$ on the torus will be in state $\hat{p}_r$, where:

$\hat{p}_r = p_r$ for $1 \le r < i$;

$\hat{p}_r = p_{[r-i]_{j-i}+i}$ for $i \le r \le l+s+2$.

In particular,

$\hat{q}_{k+1} = \hat{p}_{l+s+2} = p_{[l+s+2-i]_{j-i}+i}$

$= p_{[s+2-i]_{j-i}+i}$ (since $(j-i) \mid l = \mathrm{lcm}\{2,\ldots,s\}$)

$= q_{k+1}$. Q.E.D.

Definition. For $0 \le x_1, x_2 < w$ and $0 \le y_1, y_2 < h$:

(i) $d^1_x(x_1,x_2) = \min([x_1-x_2]_w, [x_2-x_1]_w)$.

(ii) $d^1_y(y_1,y_2) = \min([y_1-y_2]_h, [y_2-y_1]_h)$.

(iii) $d'(\langle x_1,y_1\rangle, \langle x_2,y_2\rangle) = d^1_x(x_1,x_2) + d^1_y(y_1,y_2)$.

Lemma.

(i) d' is a metric on the space $\{\langle x,y\rangle \mid 0 \le x < w, 0 \le y < h\}$.

(ii) For connector points p_1 and p_2, $d'(f(p_1),f(p_2)) \le d(p_1,p_2)$.

Finite Ribbon Theorem: Let s-state machine M begin at $p_1 \in C$ on the torus. After some number of steps of computation, suppose the connector points M has visited are, in order, $p_1,p_2,\ldots,p_r$, where the first r-1 of these points are unmarked. Then each p_i is within distance s-1 (with respect to metric d') of some point in $R'(p_1) = \{f(v) \mid v \in R(p_1)\}$.

Proof. Let q be the state in which M was started. By the Projection Theorem, M, started at p_1 in state q on the ∞ connector graph will visit a sequence of r connector points, say $p_1,p_2',p_3',\ldots,p_r'$. By the ∞ Ribbon Theorem, for each p_i' $(2 \le i \le r)$, there is a $v_i \in R(p_1)$ such that $d(p_i',v_i) \le s-1$. By (ii) of the preceding lemma, $d'(f(p_i'),f(v_i)) \le d(p_i',v_i)$. By the Projection Theorem $p_i = f(p_i')$, so $d'(p_i,f(v_i)) \le s-1$ with $f(v_i) \in R'(p_1)$. Q.E.D.

Marked Finite Ribbon Theorem: Let s-state machine M begin at $p_1 \in C$ on the torus. After some number of steps of computation, suppose the connector points M has visited are, in order, $p_1,p_2,\ldots,p_r$, where $p_2,p_3,\ldots,p_{r-1}$ are unmarked. Then each p_i is within distance s (with respect to metric d') of some point in $R'(p_1)$.

Proof. The Finite Ribbon Theorem, applied to $p_2,\ldots,p_r$, shows that for each p_i $(2 \le i \le r)$ there is $v_i \in R'(p_2)$ such that $d'(p_i,v_i) \le s-1$. By definition of R', $v_i + (p_1-p_2) \in R'(p_1)$. Then $d'(p_i,v_i+(p_1-p_2)) \le d'(p_i,v_i) + d'(v_i,v_i+(p_1,p_2)) = d'(p_i,v_i) + 1 \le s$.

Ribbon Analysis: For $0 \le x < w$ and $0 \le y < h$, each of the following sets is equal to $R'(\langle x,y\rangle)$:

(i) $\{\langle [na+x]_w,[nb+y]_h\rangle \mid n \in \mathbb{N}, a,b \in \mathbb{Z}, |a|+|b| \le s\}$

(ii) $\{\langle [\hat{n}\hat{a}+x]_w,[\hat{n}\delta+y]_h\rangle \mid \hat{n},\hat{a} \in \mathbb{N}, \delta \in \mathbb{Z}, |\hat{a}|+|\delta| \le s\}$

(iii) $H(y) \cup V(x) \cup R^1_2(\langle x,y\rangle)$, where

$H(y) = \{\langle a,y\rangle \mid 0 \le a < w\}$; $V(x) = \{\langle x,b\rangle \mid 0 \le b < h\}$; $R^1_2(\langle x,y\rangle) = \{\langle [na+x]_w,[nb+y]_h\rangle \mid n,a \in \mathbb{N}^+, b \in \mathbb{Z}-\{0\}, a+|b| \le s\}$.

(iv) $H(y) \cup V(x) \cup F(\langle x,y\rangle)$, where

$F(\langle x,y\rangle) = \{\langle [\hat{n}a+x]_w,[\hat{n}b+y]_h\rangle \mid \hat{n},a \in \mathbb{N}^+, b \in \mathbb{Z}-\{0\}, 0 \le \hat{n}a < w, a+|b| \le s\}$.

Proof. R' = (i): This is a restatement of the definition of R' given in the Finite Ribbon Theorem.

(ii) equals (i): An arbitrary $\langle [na+x]_w,[nb+y]_h\rangle$ from set (i) with $a < 0$ must be expressed in the form $\langle [\hat{n}\hat{a}+x]_w,[\hat{n}\delta+y]_h\rangle$ of set (ii) where $\hat{a}$ is required to be non-negative. The values $\hat{n} = n(wh-1)$, $\hat{a} = -a$, $\delta = -b$ are appropriate:

$\langle [na+x]_w,[nb+y]_h\rangle$

$= \langle [na+x]_w,[nb+y]_h\rangle + \langle [nwh(-a)]_w,[nwh(-b)]_h\rangle$

(since the second term is $\langle 0,0\rangle$)

$= \langle [n(wh-1)(-a)+x]_w,[n(wh-1)(-b)+y]_h\rangle$

$= \langle [\hat{n}\hat{a}+x]_w,[\hat{n}\delta+y]_h\rangle$.

(iii) equals (ii): H(y) and V(x) separate out the cases where, in (ii), $\hat{n}\delta = 0$ and $\hat{n}a = 0$, respectively.

(iv) equals (iii): To show: $R'_2(\langle x,y\rangle) \subseteq F(\langle x,y\rangle)$, the other containment being easy. Let $\langle [na+x]_w,[nb+y]_h\rangle \in R^1_2(\langle x,y\rangle)$. Pick $q \in \mathbb{N}^+$ such that $na = q*w + [na]_w$. Since $0 < a \le s$, $a \mid w = h*\mathrm{lcm}\{2,3,\ldots,s\}$, and this implies $a \mid [na]_w$. Therefore $na = q*w + \hat{n}a$ with $\hat{n} \in \mathbb{N}$ and $\hat{n}a < w$. Then

$\langle [na+x]_w,[nb+y]_h\rangle$

$= \langle [q*w+\hat{n}a+x]_w,[((q*w+\hat{n}a)/a)b + y]_h\rangle$

$= \langle [\hat{n}a+x]_w,[\hat{n}b+y]_h\rangle$

(equality holding in the second coordinate because $a*h \mid w$).

Isolation Lemma 1: Let s-state machine M be started at $\langle 0,0\rangle$ on the marked torus. Suppose M visits the sequences of connector points $p_1 = \langle 0,0\rangle, p_2,\ldots,p_r$, where p_1 and p_r are marked, and $p_2,\ldots,p_{r-1}$ are unmarked. Then $p_r = \langle 0,0\rangle$.

Proof. By the Marked Finite Ribbon Theorem, each p_i is within distance' s of R'(<0,0>). Therefore it suffices to show:

Sublemma 1: Marked vertex $m_2 = \langle s+1, 2s^2 \rangle$ is not within distance' s of R'(<0,0>).

Proof of Sublemma: By characterization (iv) of the Ribbon Analysis,

$$R'(\langle 0,0\rangle) = H(0) \cup V(0) \cup F(\langle 0,0\rangle)$$

where $F(\langle 0,0\rangle) = \{\langle na, [nb]_h \rangle \mid n,a \in \mathbb{N}^+, b \in \mathbb{Z}-\{0\}, 0 \leq na < w, a+|b| \leq s\}$.

m_2 is easily shown to be at distance' s+1 from V(0) and distance' $2s^2$ from H(0). It remains to show: $d'(m_2,p) > s$ for $p \in F(\langle 0,0\rangle)$. Suppose, to the contrary, that there is $\langle na,[nb]_h\rangle \in F(\langle 0,0\rangle)$ such that

(*) $d'(\langle s+1,2s^2\rangle,\langle na,[nb]_h\rangle)$

$$= d'_x(s+1,na) + d'_y(2s^2,[nb]_h) \leq s$$

Since $d'_x(s+1,na) \leq s$, $1 \leq na \leq 2s+1$. Therefore $0 < n \leq 2s+1$. There are now 2 cases.

Case 1: $b > 0$. Since $0 < b \leq s-1$, $0 < nb \leq 2s^2-s-1$. But this means that

$$d'_y(2s^2,[nb]_h) = \min([2s^2-[nb]_h]_h,[[nb]_h-2s^2]_h)$$

(where $h = 4s^2$)

$$\geq \min([2s^2-(2s^2-s-1)]_h,[(2s^2-s-1)-2s^2]_h)$$

$$= [2s^2-(2s^2-s-1)]_h,$$

$$= s+1$$

(since the first term is always smaller), contradicting (*).

Case 2: $b < 0$. Therefore $0 < -b \leq s-1$, which implies $0 \leq -nb \leq 2s^2-s-1$. Then

$$d'_y(2s^2,[nb]_h) = \min([2s^2-[nb]_h]_h,[[nb]_h-2s^2]_h)$$

$$\geq [-4s^2+s+1]_h = s+1$$

(since the second term is always smaller), contradicting (*). This concludes the proof of Case 2, the Sublemma, and Isolation Lemma 1.

Corollary. Let s-state machine M be started at the marked connector point <0,0>. M will never appear at marked connector point m_2 during the ensuing computation.

Proof: If M were to appear at M_2, there would be a segment of the computation where M starts at <0,0>, visiting no other marked connected points until m_2 is reached. This violates Isolation Lemma 1.

Isolation Lemma 2: Let s-state machine M be started at m_2 on the marked torus. Suppose M visits the sequence of connector points $p_1 = m_2, p_2, \ldots, p_r$, where p_1 and p_r are marked and $p_2, \ldots, p_{r-1}$ are unmarked. Then $p_r = m_2$.

Proof. By the Marked Finite Ribbon Theorem each p_i is within distance' s of $R'(m_2)$. Therefore it suffices to show:

Sublemma 2: Marked vertex <0,0> is not within distance' s of $R'(m_2)$.

Sublemma 2 follows from Sublemma 1, together with:

Symmetry Lemma: Let $0 \leq x_1, x_2 < w$ and $0 \leq y_1, y_2 < h$. Suppose $\langle x_1, y_1\rangle$ is at distance' d from some $p_2 \in R'(\langle x_2, y_2\rangle)$. Then $\langle x_2, y_2\rangle$ is at distance' d from some $p_1 \in R'(\langle x_1, y_1\rangle)$.

Proof of Symmetry Lemma: By characterization (i) of the Ribbon Analysis, there are $n \in \mathbb{N}$, $a,b \in \mathbb{Z}$ with $|a|+|b| \leq s$ such that $p_2 = \langle [na+x_2]_w, [nb+y_2]_h\rangle$. By hypothesis,

$$d = d'(\langle x_1,y_1\rangle,\langle [na+x_2]_w,[nb+y_2]_h\rangle)$$

$$= \min([x_1 - [na+x_2]_w]_w,[[na+x_2]_w-x_1]_w)$$

$$+ \min([y_1 - [nb+y_2]_h]_h,[[nb+y_2]_h-y_1]_h)$$

(definition of d')

$$= \min([[x_1+n(-a)]_w-x_2]_w,[x_2-[x_1+n(-a)]_w]_w)$$

$$+ \min([[y_1+n(-b)]_h-y_2]_h,[y_2-[y_1+n(-b)]_h]_h)$$

$$= d'(p_1,\langle x_2,y_2\rangle)$$

where $p_1 = \langle [x_1+n(-a)]_w, [y_1+n(-b)]_h\rangle$. Since $|(-a)|+|(-b)| = |a|+|b| \leq s$, $p_1 \in R'(\langle x_1,y_1\rangle)$ by characterization (i) of the Ribbon Analysis. Q.E.D.

Corollary to Isolation Lemma 2: Let s-state machine M be started at the marked connector point m_2. M will never appear at the marked point <0,0> during the ensuing computation.

Proof: As for the Corollary to Isolation Lemma 1.

The Corollaries to Isolation Lemmas 1 and 2 are clearly equivalent to the Isolation Theorem. The proof of the Isolation Theorem is now complete.

References

[Blu] Blum, M. & Hewitt, C., "Automata on a 2-dimensional tape," IEEE Conference Record, 8th Annual Symposium on Swtiching and Automata Theory, 1967, 155-160.

(Conway, J H.)
[Con] Mathematical Games, Scientific American, October 1970, 120-123; February 1971, 112-117.

[Coo] Cook, S., in preparation.

[Min] Minsky, M. L. Computation: Finite and Infinite Machines, Englewood Cliffs, N.J.: Prentice Hall, 1967.

[Pat] Beeler, M., "Paterson's Worms," AI Memo No. 290, M.I.T., June 1973.

[Rac] Rackhoff, C., in preparation.

[Sip] Sipser, M., personal communication, 1977.