APPLICATIONS OF AUTOMATA THEORY

TO THE DESIGN OF INTELLIGENT MACHINES

by

William John Sakoda

Memorandum No. UCB/ERL M78/33

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Applications of Automata Theory to the Design of Intelligent Machines

By

William John Sakoda

A.B. (Harvard University) 1972

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Approved:

Committee in Charge

Applications of Automata Theory to the Design of Intelligent Machines

by

William John Sakoda

Applications of Automata Theory to the Design of Intelligent Machines

William John Sakoda

Abstract

The computational complexity of inferring grammars for various classes of languages from finite samples from the languages is considered in the two papers comprising section 1. Section 2 discusses the capability of cooperating finite state machines moving on 2- and 3-dimensional obstructed checkerboards. Each paper is self contained and may be read independently.

Section 1.1 develops a class of inductive inference machines called adaptive recognizers. An adaptive recognizer R is presented a sequence of facts about a language L. Each fact is of the form "string x is (or is not) in L". R reads this information, maintaining a conjecture as to the language being presented. A finite sequence of facts which causes R to settle upon a correct conjecture is called an R-primer for L. R is said to learn L if such a primer exists.

Theorem 1 characterizes the language families $\mathcal{L}$ which can be learned by adaptive recognizers: The languages learned by an adaptive recognizer are always a subset of some language family $\mathcal{L}$ having an effectively enumerable sequence of decidable grammars. Conversely, if there is an effective enumeration of

decidable grammars for $\mathcal{L}$, then there is an adaptive recognizer which learns every language in $\mathcal{L}$.

Hence, there is an adaptive recognizer which learns every context-free language.

The length of a shortest R-primer for the language L is a measure of how much information R needs to identify L.

Theorem 2: Let G_1, G_2, ... be an r.e. sequence of decidable grammars, and R any adaptive recognizer learning every G_i. The size of R-primer for G_i is bounded above by a recursive function f(i) iff it is effectively decidable, given i and j, whether $L(G_i) = L(G_j)$.

Since this equivalence problem is undecidable for the linear context-free languages, the size of R-primer for linear context-free language L(G) is not bounded above by any recursive function of G.

Section 1.2 considers a model of grammatical inference where the sample strings are generated during a dialog between learning and teaching machines. Teacher t is given a grammar G as an input. The learner can pose to the teacher questions of the form "Is string x in L(G)?" . The teacher answers such questions. The teacher may also select strings x_1, x_2, ..., x_k, informing the learner whether or not each x_i is in L(G). Teacher t is said to have

Acknowledgements

I am deeply indebted to Manuel Blum for guiding my graduate studies.

The assistance of Ruth Suzuki and Vianne Ramirez in the preparation of this manuscript is gratefully acknowledged.

Table of Contents

conveyed L(G) to learner ℓ when ℓ correctly hypothesises a grammar generating L(G) and never changes the hypothesis thereafter. A pair of theorems shows that regular languages can be conveyed more quickly than linear context-free languages.

Theorem 1: The number of steps of computation required for a teacher to convey to a learner an arbitrary linear context-free language L(G) is not bounded above by any recursive function of G.

Theorem 4: There is a teacher t which, given an n-state deterministic finite automaton M, conveys L(M) to a certain learner ℓ. The number of steps of computation required to convey L(M) is bounded above by a polynomial p(n).

Section 2 considers cooperating finite automata moving on 2- and 3-dimensional obstructed checkerboards. In 2 dimensions, 1 machine with 4 pebbles can search every finite 2-dimensional obstructed checkerboard. One machine with 7 pebbles suffices to search infinite 2-dimensional space. In contrast, no finite number of finite automata can search every finite 3-dimensional obstructed checkerboard.

Acknowledgement

Research sponsored by the National Science Foundation Grant MCS75-21760.

§1.1 ADAPTIVE LANGUAGE RECOGNITION

Introduction

There are numerous interesting families of formal languages (e.g. regular, context-free) for which the problem of identification in the limit admits a solution by enumeration. In this report we ask whether the task of identifying such families may be accomplished more efficiently.

A class of inductive inference machines, the adaptive recognizers, is developed and used to analyze the complexity of inferring grammars for languages. The classes of languages which can be inferred by adaptive recognizers are characterized in Theorem 1. Theorem 2 shows that if the equivalence problem for a sequence L_1, L_2,... of languages is undecidable, then the amount of data consumed by an adaptive recognizer before correctly identifying the language L_i is not bounded above by any recursive function of i.

1.1.1. Preliminary Notation

We will use the symbol $\mathbb{N}$ to denote the set of natural numbers; that is, $\mathbb{N} = \{0,1,2,\ldots\}$. An object will be said to be an integer if and only if it is a member of $\mathbb{N}$. A language is a subset of $\mathbb{N}$.

We assume familiarity with some concepts from elementary recursive function theory. Our notation for recursive functions will follow that in [6].

1.1.2. Definition of the Model

Adaptive recognizers are machines which attempt to identify languages on the basis of finite samples from the languages. They differ from the usual rule inference models [1,4] in their mode of identification: an adaptive recognizer demonstrates its identification of a language by performing as a recognizer for that language.

Before proceeding, some notation for manipuating sequences will be useful.

1.1.2.1. Definition

(1) Let $x_1,x_2,\ldots,x_n$ be integers, with $n \geq 0$. Then $\langle x_1,x_2,\ldots,x_n\rangle$ denotes an integer which encodes the ordered sequence with elements $x_1,x_2,\ldots,x_n$, according to some fixed encoding.(1) We will usually use vectored variables to range over such sequence numbers. For

$$\vec{s} = \langle x_1,\ldots,x_n\rangle$$

and

$$\vec{t} = \langle y_1,\ldots,y_m\rangle ,$$

we let

$$\vec{s}\cdot\vec{t} = \langle x_1,\ldots,x_n,y_1,\ldots,y_m\rangle .$$

(2) Let $n \in \mathbb{N}$. Let $x_1,\ldots,x_n \in \mathbb{N}$. Let $b_1,\ldots,b_n \in \{0,1\}$. Then $\vec{s} = \langle\langle x_1,b_1\rangle,\langle x_2,b_2\rangle,\ldots,\langle x_n,b_n\rangle\rangle$ is a sample from language L if and only if, for each $1 \leq i \leq n$,

$$b_i = \begin{cases} 0 & \text{if } x_i \notin L \\ 1 & \text{if } b_i \in L . \end{cases}$$

The set $\{x_i \mid 1 \leq i \leq n\}$ is the base of the sample $\vec{s}$. The size of sample $\vec{s}$, denoted $sz(\vec{s})$, is defined to be $\max(\{0\} \cup \{x_i \mid 1 \leq i \leq n\})$. The length of sample $\vec{s}$ is the cardinality of the base of $\vec{s}$.

(3) $S = \{\vec{s} \mid (\exists \text{ language L})$ such that $\vec{s}$ is a sample from L$\}$. S is the set of all samples. Note that S is recursive.

We can now indicate the difference between adaptive recognizers and the usual rule-inference machines. Let $\vec{s}$ be a sample from language L_o. A rule-inference machine would use $\vec{s}$ to attempt to produce a name (say, a partial recursive index) for L_o. An adaptive recognizer uses $\vec{s}$ to attempt to function as a recognizer for L_o. Thus, given $\vec{s}$ and an integer x, an adaptive recognizer generates a guess as to whether x is in the language L_o.

1.1.2.2. Definition

An adaptive recognizer is a function $r: S\times\mathbb{N} \to \{0,1\}$ such that:

(a) r is partial recursive, with $dom(r) = S\times\mathbb{N}$;

and (b) (permutation independence): If $\vec{s}$, $\vec{t}$ are samples and $\vec{s}$ is

a permutation of $\vec{t}$, then $r(\vec{s},x) = r(\vec{t},x)$, for all $x \in \mathbb{N}$.

The restriction of permutation independence is made to force r to base its guesses entirely on the membership information contained in sample $\vec{s}$, and not on the order in which this information appears in $\vec{s}$.

1.1.2.3. Definition

$\vec{s} \in S$ is an r-primer for language L if, for any sample $\vec{e}$ from the language L and any $x \in \mathbb{N}$,

$$r(\vec{s}\cdot\vec{e},x) = \begin{cases} 0 & \text{if } x \notin L \\ 1 & \text{if } x \in L \end{cases}.$$

r is said to recognize (equivalently, identify) L if there exists an r-primer for L. R_r is by definition the set of languages recognized by r.

Since primers are samples, they inherit the notion of size defined for samples in 1.1.2.1(2). The size of a smallest r-primer for a particular language L is a useful measure of the amount of data required by r to identify L.

1.1.3. Recursively Indexed Families of Languages

Our interest in adaptive recognizers arises from their close connection with recursively indexed families of languages. Intuitively, we want the recursively indexed families to encompass exactly the classes of languages which can be identified in the limit by enumeration.

1.1.3.1. Definition

A family F of languages is said to be recursively indexed if there is a recursive $\psi: \mathbb{N} \to \mathbb{N}$ which enumerates at least one index for the characteristic function of each language in F, and only such indices.

Whenever ψ is a recursive indexing, we let $L_i^{\psi} = \{x \in \mathbb{N} \mid \phi_{\psi(i)} = 1\}$. Thus, L_i^{ψ} is that language for which $\phi_{\psi(i)}$ is the characteristic function.

1.1.3.2. Examples of Recursively Indexed Families

(i) The class of languages having primitive recursive characteristic functions is recursively indexed.

(ii) By establishing a suitable identification between $\mathbb{N}$ and $(0,1)^*$, the notion of recursive indexing may be extended to families of subsets of $(0,1)^*$.[2]

Assuming that such an identification has been established, we can then say that the families of regular, context-free, and context-sensitive subsets of $(0,1)^*$ are each recursively indexed. In each of these families, a recursive indexing can be obtained by using the fact that there is a recursive enumeration $G_0, G_1, G_2, \ldots$ of Gödel numbers for the class of grammars in question, such that the predicate $x \in L(G_i)$ is decidable uniformly in x and i.

(iii) More generally, if L is a recursive subset of $(0,1)^*$, the family of subsets of $(0,1)^*$ in the principal AFL generated by L can be recursively indexed.

1.1.4. Summary of Results

Our first result is a characterization of the families of languages which can be identified by an adaptive recognizer.

Theorem 1. Let F be a family of languages. Then the following are equivalent.

1) There is an adaptive recognizer which can identify at least every language in F.

2) F is contained in a recursively indexed family.

3) Every language in F is h-easy[3] for some fixed recursive $h: \mathbb{N} \to \mathbb{N}$.

The second result yields a condition on a recursively indexed family F which is sufficient to rule out the existence of an efficient adaptive recognizer identifying every language in F. Some notation is necessary before proceeding.

1.1.4.1. Definition

Let ψ be a recursive indexing. The equivalence problem for ψ is that function $e: \mathbb{N} \times \mathbb{N} \to \{0,1\}$ which is defined by

$$e(i,j) = \begin{cases} 0 , & L_i^{\psi} \neq L_j^{\psi} \\ 1 , & L_i^{\psi} = L_j^{\psi} . \end{cases}$$

The equivalence problem for ψ is decidable just in case e is recursive.

Theorem 2'. Let $\psi: \mathbb{N} \to \mathbb{N}$ recursively index family F, and suppose the equivalence problem for ψ is undecidable. Let r be any

adaptive recognizer which identifies at least the language in F. Let $\|L_i^\psi\|$ be the size of a smallest r-primer for L_i^ψ. Then $\|L_i^\psi\|$ is not bounded above any any recursive function $m(i)$.

Note that if the restriction that adaptive recognizers be permutation independent is removed, the theorem fails rather dramatically: for any recursive indexing ψ, there is a non-permutation-independent r such that $\|L_i^\psi\| = 1$ for every i! An algorithm for such an r is: "On input $(\langle\langle x_1,b_1\rangle,\ldots\rangle,x)$, output 1 if $x \in L_{x_1}^\psi$, 0 otherwise." Then $\langle\langle i,\phi_{\psi(i)}(i)\rangle\rangle$ is an r-primer for L_i^ψ.

Corollary - Application to the Context-Free Languages

We remarked in 1.1.3.2 that the notion of recursive indexing could be extended to languages which are subsets of $(0,1)^*$ by establishing an identification between $(0,1)^*$ and $\mathbb{N}$; note (2) indicates a suitable bijection $\wedge : (0,1)^* \rightarrowtail\!\!\!\!\!\rightarrow \mathbb{N}$. This identification also allows results about adaptive recognizers to be extended to such languages. We will outline this technique.

The following conventions are useful. For $x \in (0,1)^*$, we let $\hat{x}$ denote the image of x under the map $\wedge$. For $A \subseteq (0,1)^*$, $\hat{A} = \{\hat{x} \mid x \in A\}$. Finally, $\hat{CFL} = \{\hat{L} \mid L \text{ is a context free subset of } (0,1)^*\}$.

We can now restate the preceding definitions using this notation. It should be emphasized that we are literally repeating what was said before, using a slightly different notation.

Let $n \geq 0$; let $x_1,\ldots,x_n \in (0,1)^*$; and let $b_1,\ldots,b_n \in \{0,1\}$. Then $\vec{s} = \langle\langle\hat{x}_1,b_1\rangle,\langle\hat{x}_2,b_2\rangle,\ldots,\langle\hat{x}_n,b_n\rangle\rangle$ is a sample from language $L \subseteq (0,1)^*$ if for each $1 \leq i \leq n$,

$$b_i = \begin{cases} 0 , & x_i \notin L \\ 1 , & x_i \in L . \end{cases}$$

S, the set of all samples, is by definition $\{\vec{s} \mid \vec{s}$ is a sample from some $L \subseteq (0,1)^*\}$.

Sample $\vec{s}$ is an r-primer for $L \subseteq (0,1)^*$ if for every sample $\vec{e}$ from L and every $x \in (0,1)^*$,

$$r(\vec{s}\cdot\vec{e},\hat{x}) = \begin{cases} 0 , & x \notin L \\ 1 , & x \in L . \end{cases}$$

One modification is in order. If $\vec{s} = \langle\langle\hat{x}_1,b_1\rangle,\ldots,\langle\hat{x}_n,b_n\rangle\rangle$ is a sample from $L \subseteq (0,1)^*$, it is more natural to define the size of $\vec{s}$ to be the length of the longest string appearing in $\vec{s}$. Thus, $\hat{sz}(\vec{s})$ is by definition $\max(\{0\} \cup \{\text{length}(x_i) \mid 1 \leq i \leq n\})$.

We are now prepared to apply Theorem 2' to the context-free languages. Let ψ be that recursive indexing of $\hat{CFL}$ which is induced by the standard Gödel numbering of the context-free grammars over the terminal alphabet $\{0,1\}$; i.e. $L_i^{\psi} = \hat{L}(G_i)$, where G_i is the i^{th} standard context-free grammar. Since it is undecidable, given CFG's G_i and G_j, whether $L(G_i) = L(G_j)$, it follows that the equivalence problem for ψ is undecidable. Theorem 2' then supports the following result: Let r be any adaptive recognizer which can identify at least $\hat{CFL}$. Let $\hat{\|}G_i\hat{\|}$ denote the smallest r-primer (in the sense of $\hat{sz}$) for $L(G_i)$. Then there is no algorithm which, given an arbitrary G_i, will compute an upper bound on $\hat{\|}G_i\hat{\|}$.

1.1.5. Proofs

1.1.5.1. Characterization Theorem for Adaptive Recognizers

The following lemma records an important property of recursively indexed families. Indeed, the definition of recursive indexing was chosen specifically to ensure that this property held.

1.1.5.1.1. Lemma

Let ψ be a recursive indexing. Then there exists, uniformly effectively in ψ, a recursive characteristic function for the relation $\lambda x,i[x \in L_i^{\psi}]$.

Proof. By definition of L_i^{ψ}, $\phi_{\psi(i)}$ is the characteristic function for L_i^{ψ}. Therefore $\lambda x,i[\phi_{\psi(i)}(x)]$ is the required characteristic function.

This lemma will be used implicitly in subsequent constructions.

Theorem 1. Let F be a family of languages. Then the following are equivalent.

(1) $F \subseteq R_r$ for some adaptive recognizer r.

(2) $F \subseteq G$ for some recursively indexed family G.

(3) Every language in F is h-easy for some fixed recursive $h: \mathbb{N} \to \mathbb{N}$.

The proof that (2) → (1) involves a construction which will be of use later. We record it below.

1.1.5.1.2. Lemma

Let family F of languages be recursively indexed by ψ. Then there exists, uniformly effectively in ψ, an adaptive recognizer r with the following properties:

(1) r recognizes exactly F.

(2) For each $j \in \mathbb{N}$, the following non-effective procedure leads to a primer, $p(j)$, for L_j^ψ.

(i) Pick i least such that $L_i^\psi = L_j^\psi$.

(ii) For each $k < i$, pick an x_k such that $x_k \in L_k^\psi \leftrightarrow x_k \notin L_i^\psi$.

(iii) Let $p(j)$ be a sample from L_j^ψ with base $\{x_k \mid 1 \leq k < i\}$.

Proof. We will construct the required r.

On input $(\vec{s},m)$, r will try to find an $\vec{s}$-consistent hypothesis $L \in F$, and output 0 if $m \notin L$, 1 otherwise. Since F is recursively indexed, this search may be carried out in an orderly fashion by testing $L_0^\psi, L_1^\psi, \ldots$ for $\vec{s}$-consistency. The danger of never finding an $\vec{s}$-consistent L_i^ψ may be handled by bounding the number of L_i^ψ tested by the length ℓ of the sample $\vec{s}$.

Thus, if $L_0^\psi, \ldots, L_{\ell+1}^\psi$ are all $\vec{s}$-inconsistent, we set

$$r(\vec{s},m) = \begin{cases} 0, & m \notin L_0^\psi \\ 1, & m \in L_0^\psi \end{cases}.$$

If, on the other hand, one of $L_0^\psi, \ldots, L_{\ell+1}^\psi$ is $\vec{s}$-consistent, we pick the least n such that L_n^ψ is $\vec{s}$-consistent, setting

$$r(\vec{s},m) = \begin{cases} 0, & m \notin L_n^\psi \\ 1, & m \in L_n^\psi \end{cases}.$$

□

It's clear that r is defined on all of $S \times \mathbb{N}$ and satisfies permutation independence. Since for any $\vec{s}$, $(\lambda x[r(\vec{s},x)]) = \chi_L$ for some $L \in F$, we have $R_r \subseteq F$. Finally, to verify property (2) of the lemma, let $j \in \mathbb{N}$, and let $p(j)$ be a sample from L_j^ψ with base $\{x_k \mid 1 \leq k < i\}$ supplied by the construction indicated in (2). We need to show that $(\lambda x[r(p(j)\cdot\vec{e},x)]) = \chi_{L_j^\psi}$ for any sample $\vec{e}$ from L_j^ψ. To do this, it will certainly suffice to show that, when given $p(j)\cdot\vec{e}$ as input, r selects L_j^ψ as its hypothesis. But the information contained in $p(j)$ is sufficient to cause all L_k^ψ, $k < j$ to be rejected; the length of $p(j)$ is sufficient to allow the search to reach at least to L_j^ψ; and L_j^ψ cannot be rejected, since it is certainly consistent with $p(j)\cdot\vec{e}$. Thus $p(j)$ is a primer for L_j^ψ, and (2) has been verified. (2) now implies that $R_r \supseteq F$, which fact, combined with the reverse containment proved earlier, yields $R_r = F$.

Proof of Theorem 1. The equivalence of (2) and (3) is easily verified. We will show that (1) and (2) are equivalent.

(1) → (2). Let r be an arbitrary adaptive recognizer. We will construct a ψ which recursively indexes a superset of R_r.

Roughly speaking, what we are trying to do is establish an effective correspondence between the integers and the languages in R_r. This can be accomplished by exploiting a natural correspondence between S and R_r. The latter correspondence is the following. To each $\vec{s} \in S$, associate $X_{\vec{s}} = \{m \in \mathbb{N} \mid r(\vec{s},m) = 1\}$. This association is sufficiently effective since, given $\vec{s}$, we can find an algorithm for $\chi_{X_{\vec{s}}}$: on input y, the algorithm simply evaluates $r(\vec{s},y)$. To see that this scheme does indeed manage to assign some $\vec{s} \in S$ to every $L \in R_r$, note that

if $L \in R_r$, there is an r-primer, $\vec{p}$, for L. Then clearly $L = X_{\vec{p}}$.

We may end up indexing a proper superset of F, as unless $\vec{s}$ is a primer, there is no reason to expect that $X_{\vec{s}} \in R_r$.

To finish up, let $s: \mathbb{N} \longmapsto\!\!\!\!\rightarrow S$ be a recursive bijection between $\mathbb{N}$ and S. Then an appropriate algorithm for ψ is: "On input x, output the index of an algorithm which computes $\lambda y[r(s(x),y)]$". Then ψ is a recursive indexing, since r is 0-1 valued and convergent on $S \times \mathbb{N}$. ψ indexes at least R_r, since if $L \in R_r$ and $\vec{p}$ is an r-primer for L, then $\psi(s^{-1}(\vec{p}))$ is an index for the characteristic function of L.

(2) → (1). If G is recursively indexed, Lemma 1.1.5.1.2 supplies an r which recognizes exactly G.

1.1.5.2. The Role of the Equivalence Problem

We now turn to a proof of the second result. Roughly stated, this result is that undecidability of the equivalence problem for a recursive indexing ψ implies that for any adaptive recognizer, r, which identifies at least all the L_i^{ψ}, it is difficult to generate r-primers for ψ. Our method for measuring said difficulty will be to test for the existence of an effective procedure which, given an arbitrary integer i, will produce an r-primer for L_i^{ψ}.

1.1.5.2.1. Definition

Let ψ be a recursive indexing of family F, and let r be an adaptive recognizer identifying at least F. Then $p: \mathbb{N} \to \mathbb{N}$ is a generator of r-primers for ψ if, for all $i \in \mathbb{N}$, $p(i)$ is an r-primer for L_i^{ψ}.

We will prove a slightly stronger version of Theorem 2' of Section 1. .4, namely:

Theorem 2. Let ψ be a recursive indexing of F.

(a) Let r be an adaptive recognizer identifying at least F, and let $p: \mathbb{N} \to \mathbb{N}$ be a generator of r-primers for ψ. Then the equivalence problem for ψ is recursive in p.[4]

(b) There is an adaptive recognizer, r, such that

(i) r recognizes exactly F; and

(ii) there is a generator of r-primers for ψ which is recursive in the equivalence problem for ψ.

The following lemma is the basis for our proof of part (a) of the theorem.

1.1.5.2.2. Lemma

Primers for distinct languages are inconsistent. That is, let L, L' be languages recognized by adaptive recognizer r. Let $\vec{q}$ be an r-primer for L. Let $\vec{q}'$ be an r-primer for L'. If $L \neq L'$, then either

(*) $\vec{q}$ is not a sample from L'; or

(*') $\vec{q}'$ is not a sample from L.

Proof. Suppose the lemma is false. Then for some $L \neq L'$, both (*) and (*') fail. We will use this to get a contradiction.

Pick an $x \in \mathbb{N}$ witnessing the fact that $L \neq L'$ (say, $x \in L$ $x \notin L'$; the other case will follow by symmetry). Then

(1) $1 = r(\vec{q}\cdot\vec{q}',x)$ ($\vec{q}'$ is a sample from L since (*') is false; therefore guess $r(\vec{q}\cdot\vec{q}',x)$ must be correct, since $\vec{q}$ is a primer for L.)

(2) $= r(\vec{q}'\cdot\vec{q},x)$ (permutation independence)

(3) $= 0$ (Argue as in line 1, exchanging primed and unprimed variables everywhere.)

Contradiction, as required.

Corollary to Lemma (criterion for language equivalence). Let the notation be as in the lemma. Then

$$L = L' \leftrightarrow (\vec{q} \text{ is a sample from } L' \text{ and } \vec{q}' \text{ is a sample from } L) .$$

Proof of Corollary. $\rightarrow$ is obvious.

$\leftarrow$ is the contrapositive of the implication of the lemma.

Armed with this corollary, its now easy to prove the theorem.

Proof of Theorem 2.

(a) To decide recursively in p, given integers i, j, whether $L_i^{\psi} = L_j^{\psi}$, proceed as follows:

(1) Using the oracle for p, compute

$$\vec{q} = p(i)$$
$$\vec{q}' = p(j) .$$

(2) By the corollary to the lemma,

$$L_i^{\psi} = L_j^{\psi} \leftrightarrow (\vec{q} \text{ is a sample from } L_i^{\psi} \text{ and } \vec{q}' \text{ is a sample from } L_j^{\psi}) .$$

Thus, if we could effectively test whether the right-hand side of the

equivalence held, we would be done. It is clear that if we had a method for evaluating the characteristic functions of L_i^{ψ} and L_j^{ψ}, this test could be performed. Such a method is indeed available, since

$$\lambda x[r(\vec{q},x)] = \chi_{L_i^{\psi}}$$

and

$$\lambda x[r(\vec{q}',x)] = \chi_{L_j^{\psi}}.$$

(b) Let r be the adaptive recognizer supplied by Lemma 1.1.5.1.2. Property (1) of the lemma implies that requirement (i) of the theorem is satisfied. Given an oracle for the equivalence problem for ψ, the non-effective procedure (2) of the lemma for generating r-primers for ψ becomes effective, thus establishing claim (ii) of the theorem.

Corollary to Theorem 2 (Theorem 2' of Section 1.1.3). Let $\psi: \mathbb{N} \to \mathbb{N}$ recursively index family F, and suppose the equivalence problem for ψ is undecidable. Let r be any adaptive recognizer which identifies at least the languages in F. Let $\|L_i^{\psi}\|$ be the size of a smallest r-primer for L_i^{ψ}. Then $\|L_i^{\psi}\|$ is not bounded above by any recursive function $m(i)$.

Proof of Corollary. Suppose, to the contrary, that there is a recursive $m(i)$ which bounds $\|L_i^{\psi}\|$ from above. We will use this to get a contradiction.

The point is that using m, we can construct a recursive generator, p, of r-primers for ψ: On input i, p simply outputs a sample from L_i^{ψ} with base $\{x \mid x \leq m(i)\}$. We must verify that $p(i)$ is a primer for

L_i^{ψ}. Since $m(i)$ is an upper bound on the integers appearing in the base of some r-primer for L_i^{ψ}, there is an r-primer $\vec{q}_i$ for L_i^{ψ} and a sample $\vec{f}_i$ from L_i^{ψ} such that $p(i)$ is a permutation of $\vec{q}_i \circ \vec{f}_i$. Then

$$r(p(i)\circ\vec{e},x) = r(\vec{q}_i\circ(\vec{f}_i\circ\vec{e}),x) \quad \text{(permutation independence)}$$

$$= \begin{cases} 0\ , & x \notin L_i^{\psi} \\ 1\ , & x \in L_i^{\psi} \end{cases} \quad \text{(since } \vec{q}_i \text{ is an r-primer for } L_i^{\psi}\text{)}\ .$$

This being true for arbitrary $i \in \mathbb{N}$, p is a recursive generator of r-primers for ψ.

Now by part (a) of Theorem 2, the equivalence problem for ψ is recursive in p. But as p is recursive, this means that the equivalence problem for ψ is recursive outright, contradicting the undecidability of the equivalence problem for ψ.

1.1.6. Connections with Other Models of Inductive Inference

We will conclude by indicating a connection with a rule-inference model.

1.1.6.1. Definition

(1) A rule inference machine is a function $M: S \to \mathbb{N}$ such that

(i) M is partial recursive with $dom(M) = S$; and

(ii) for all $\vec{s} \in S$, $\phi_{M(\vec{s})}$ is 0-1 valued and defined everywhere.

Thus, $\phi_{M(i)}$ is always the characteristic function of some subset of $\mathbb{N}$.

(2) Rule inference machine M is permutation-independent if for any $\vec{s}, \vec{t} \in S$ such that $\vec{s}$ is a permutation of $\vec{t}$, $\phi_{M(\vec{s})} = \phi_{M(\vec{t})}$.

(3) Let $L \subseteq \mathbb{N}$ and let $\vec{s}$ be a sample from L. $\vec{s}$ is an M-primer for matching L [5] if for every sample $\vec{e}$ from L, $\phi_{M(\vec{s}\cdot\vec{e})} = \chi_L$. M is said to match L if and only if there is an M-primer for matching L. The notion of a generator of M-primers is defined in the obvious way.

The following proposition establishes the connection between adaptive recognizers and permutation-independent rule-inference machines.

1.1.6.2. Proposition

(a) Let M be a permutation independent rule-inference machine. Then there is an adaptive recognizer r such that for any $L \subseteq \mathbb{N}$ and any sample $\vec{s}$ from L, $\vec{s}$ is an M-primer for matching L if and only if $\vec{s}$ is an r-primer for L.

(b) Conversely, let r be any adaptive recognizer. Then there is a permutation independent rule inference machine M such that for any $L \subseteq \mathbb{N}$ and any sample $\vec{s}$ from L, $\vec{s}$ is an r-primer for L if and only if $\vec{s}$ is an M-primer for matching L.

Proof. (a) $r(\vec{s},x) = \phi_{M(\vec{s})}(x)$ is suitable.

(b) On input $\vec{s}$, M outputs the index of the following program: "On input x, output the value $r(\vec{s},x)$."

Thus, Theorems 1 and 2 may be applied to permutation-independent rule-inference machines by replacing:

"adaptive recognizer"	by	"permutation-independent rule-inference machine"
"r"	by	"M"
"r-primer"	by	"M-primer" .

Footnotes

(1) For example, the following is suitable. Let p_i denote the i^{th} prime. Let $\langle\ \rangle = 1$ and, for $n \geq 1$, let $\langle x_1,\ldots,x_n\rangle = \prod_{i=1}^{n} p_i^{(x_i+1)}$

(2) The following map $\wedge: (0,1)^* \mapsto\!\!\!\!\rightarrow \mathbb{N}$ works. Let $<$ be a total ordering of $(0,1)^*$ in order of increasing length of strings, ties between strings of equal length being broken by a lexicographic ordering. For $x \in (0,1)^*$, let $\wedge(x)$ be the number of strings preceding x in the $<$ ordering.

(3) Total recursive function ϕ_i is h-easy if $\Phi_i(x) \leq h(x)$ for almost all $x \in \mathbb{N}$, where $\Phi_i(x)$ denotes the running time of ϕ_i on input x. See [2,5] for details.

(4) For $f, g: \mathbb{N} \to \mathbb{N}$, f is <u>recursive in</u> g if, given an "oracle" for computing the function g, f may be computed effectively. See [6] for details.

(5) The notion of matching is due to Feldman [3].

References

[1] L. Blum and M. Blum, "Inductive Inference: A Recursion Theoretic Approach," Memorandum ERL-M386, Electronics Research Laboratory, University of California, Berkeley, March 1973.

[2] M. Blum, "A Machine-Independent Theory of the Complexity of Recursive Functions," J. ACM 14 (2), April 1967, 322-336.

[3] J.A. Feldman, "Some Decidability Results on Grammatical Inference and Complexity," Artificial Intelligence Memorandum 93.1, Computer Science Department, Stanford University, May 1970.

[4] E. Mark Gold, "Language Identification in the Limit," Information and Control 10, 447-474.

[5] J. Hartmanis and J.E. Hopcroft, "An Overview of the Theory of Computational Complexity," J. ACM 18 (3), July 1971, 444-475.

[6] H. Rogers, Jr., Theory of Recursive Functions and Effective Computability, McGraw-Hill, New York, 1967.

§1.2. AN APPLICATION OF THE THEORY COMPLEXITY TO INDUCTIVE INFERENCE

1.2.1 Introduction

This section considers the computational complexity of the grammatical inference problem. Previous theoretical work [2,3] has concentrated on characterizing those classes of formal languages which are inferable, without regard to the computational complexity of the inference algorithms used. Some of the more concrete work [4,7] has in fact focused on constructing efficient inference programs; however, efficiency has generally been obtained through use of complicated heuristics, which are not particularly amenable to analysis. Our goal is to narrow the gap between such efforts.

Our plan is as follows. In section 1.2.2 we will define our model of inference; as the model is new, we will give a careful sketch of it. Section 1.2.3 presents some general properties of the model. Section 1.2.4 describes a polynomial-time algorithm for the class of Regular languages, and indicates how the method may be extended to the class of recognizable trees. Finally, in section 1.2.5 we consider an operation on languages which may be used to expand the class of languages which can be recognized by inference algorithms for the class of Regular sets. The impact of our results on the possibility of efficiently inferring various families of languages in the Chomsky hierarchy is summarized below.

Language Class	Representation of Language L	Size Measure	Time required for stable convergence as a function of size measure
Regular	state graph of a deterministic Moore automaton accepting L	number of states	polynomial
R_C^k (any k)	state graph of a deterministic Moore automaton accepting L_S^k	number of states	polynomial
Even linear context-free	restricted even linear grammar generating L	number of non-terminals	polynomial
Linear context-free	linear CFG generating L	total length of right hand sides of rules	no recursive upper bound

1.2.2 An Interactive Model of Inductive Inference

In investigations of inferability where the complexity of the algorithms used for inference is not at issue, results tend to be relatively insensitive to the order in which the language is presented to the learner. Such is no longer the case when we are interested in the running times of the learning algorithms: the time required for learning will depend critically on the order in which data about the language are presented. In a sense, efficient learning rests on the learner's ability to cause the presentation of data in a useful order.

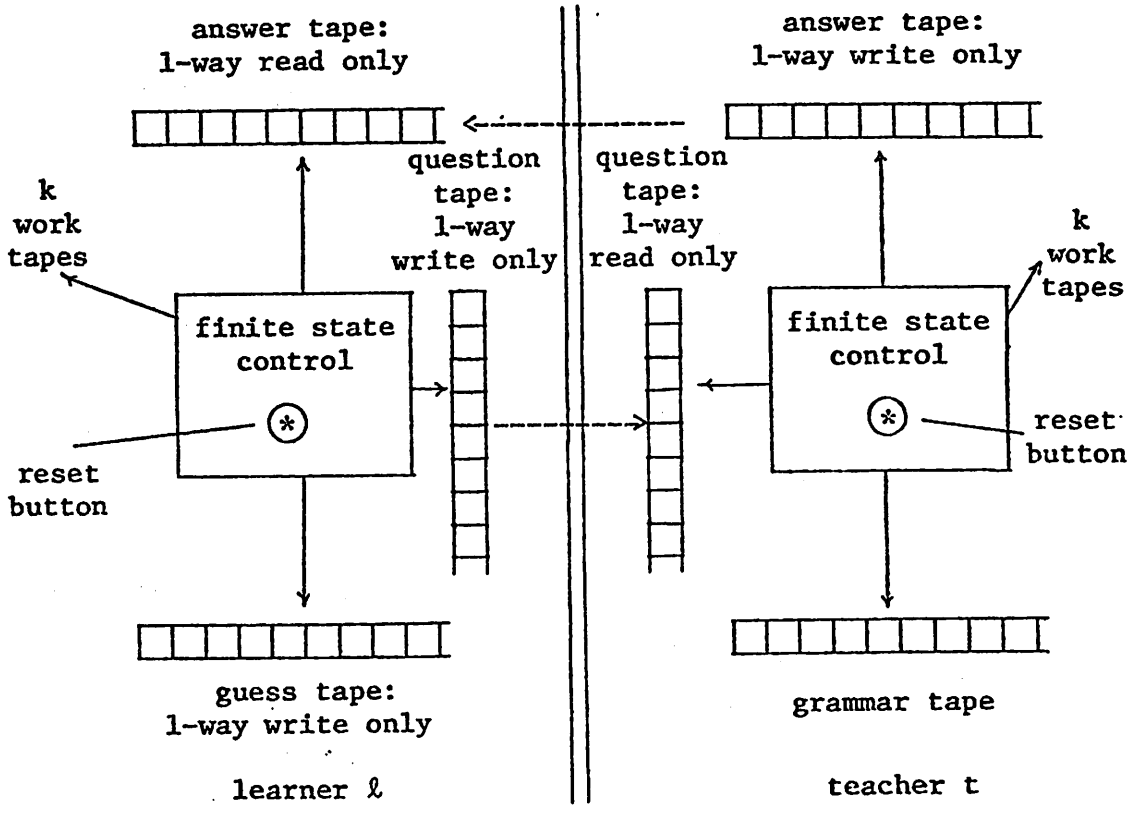
This leads us to consider a system consisting of a learner which is attempting to generate a grammar for a language, together with a teacher, which is the source of information for that language. The learner receives information about the language by presenting particular strings to the teacher and asking whether or not the strings are in the language. The teacher answers all such questions and in addition may supply information about other strings which it thinks will be helpful to the learner. Thus, both teacher and learner take an active role in attempting to speed the learning process by influencing the order of presentation of data. We feel that this characteristic is an important feature of the model, as such interaction appears to be a factor in the acquisition of natural languages, and in other learning situations relevant to work in artificial intelligence.

Before proceeding, we define the families of languages to be considered.

Definition. Let Σ be a finite alphabet, F a family of languages over Σ^*, and G a countable collection of finitary objects. A map

L: $G \twoheadrightarrow F$ ($\twoheadrightarrow$ denoting onto) is said to be a recursive indexing of F by G if it is recursively decidable, given $G \in G$ and $x \in \Sigma^*$, whether $x \in L(G)$.

In order to assess the difficulty of inferring a particular recursively indexed family L: $G \twoheadrightarrow F$, we consider systems of the sort depicted below.



Both learner ℓ and teacher t are multi-tape Turing machines equipped with a feature called a restart button. A restart button on a machine M may be used as follows. When M halts, we may replace any number of M's tapes with other tapes, possibly with non-blank symbols written on them. Pressing the restart button on M then causes M to resume operation at its initial control state. Any tapes not replaced

retain their former contents and positioning upon restarting. The restart feature will allow teacher and learner to communicate by having an operator shuffle tapes between the two machines.

To start things off we pick a $G \in \mathcal{G}$ and place an encoding[1] on t's grammar tape. ℓ then attempts to learn $L(G)$ from t by addressing questions to t. The dialogue proceeds in stages, as follows:

Stage n $(\forall n \geq 0)$

1n) ℓ picks a string from Σ^* (call it q_n) and asks: "Is q_n in $L(G)$?" The question is asked by writing q_n onto the question tape. ℓ then halts.

2n) We now remove the question tape from ℓ, install it as the question tape on t, and restart t.

3n) t must now read the question asked of it and respond by printing an encoding of the answer "q_n is (or is not, as is appropriate) in $L(G)$" on its answer tape. t may, in addition, choose other strings and inform ℓ of whether or not they are in $L(G)$ by printing the appropriate information on the answer tape. Upon completion, t halts.

4n) t's answer tape is removed and installed as the answer tape on ℓ. A blank question tape is installed on ℓ, and a blank answer tape is installed on t. ℓ is then restarted.

5n) ℓ reads the answer and, after a period of computation, prints on its guess tape G_n, its current guess at the grammar being presented to it. Stage 5n is defined as ending when the last symbol of G_n is printed.

Note that in general the answers generated by t will depend not only on the question posed, but also on the preceding sequence of questions.

We let $A_t^G(\langle q_0,\ldots,q_n\rangle)$ denote the set of answers generated by t when question q_n is asked in the context of preceding questions $q_0,q_1,\ldots,q_{n-1}$ and grammar G. If t and t' are two teachers, t' is said to <u>extend</u> t if, for every $G \in \mathcal{G}$ and any sequence of questions $\langle q_0,\ldots,q_n\rangle$, $A_{t'}^G(\langle q_0,\ldots,q_n\rangle) \supseteq A_t^G(\langle q_0,\ldots,q_n\rangle)$.

<u>Definitions</u>

(i) The system (ℓ,t) <u>conveys</u> G in q questions (with respect to the indexing L, when this is not understood) if, in the presentation of G described above, all guesses G_n for $n \geq q$ are identical, and $L(G_q) = L(G)$.

(ii) (ℓ,t) <u>stably conveys</u> G in q questions (with respect to L) if, for every t' extending t, (ℓ,t') conveys G in q questions.

(iii) Define $Q_{(\ell,t)}: \mathcal{G} \to \mathbb{N} \cup \{\infty\}$ by

$$Q_{(\ell,t)}(G) = \begin{cases} \text{the least integer q such that } (\ell,t) \text{ stably} \\ \quad \text{conveys G in q questions, if such a q exists;} \\ \infty, \text{ ——} \end{cases}$$

(iv) Define $S_{(\ell,t)}: \mathcal{G} \to \mathbb{N} \cup \{\infty\}$ by

$$S_{(\ell,t)}(G) = \begin{cases} \text{the number of atomic steps executed by the} \\ \quad \text{machines t and } \ell \text{ during the first } Q_{(\ell,t)}(G) \\ \quad \text{stages of dialogue, if } Q_{(\ell,t)}(G) < \infty; \\ \infty, \text{ ——} \end{cases}$$

The restriction of stability is made to rule out the following sort of collaboration. Consider a teacher t which, when given a grammar G and asked whether a string x is in $L(G)$, answers the question and in addition, volunteers information about exactly one other string, that string being an encoding of G. Then by asking a single question, a

learner can obtain this encoding and converge to the correct grammar.

1.2.3. General Properties of the Model

Our method for assessing the difficulty of inferring a recursively indexed family will be to consider the rate of growth of the function $S_{(\ell,t)}$ for various systems (ℓ,t). This section establishes two general results concerning this rate of growth.

Definition. Let $L: G \twoheadrightarrow F$ be a recursive indexing. The equivalence problem for L is said to be decidable if and only if there is an effective procedure for deciding, given G, $G' \in G$, whether $L(G) = L(G')$.

Theorem 1. Suppose that $L: G \twoheadrightarrow F$ is a recursive indexing with an undecidable equivalence problem, and that (ℓ,t) is a system which stably conveys every grammar in G. Then $S_{(\ell,t)}(G)$ is not bounded above by any effective computable function $R: G \to \mathbb{N}$.

Our interest in this theorem arises from the fact that many well-known families of languages have an undecidable equivalence problem with respect to their natural indexings. For example, in examining the Chomsky hierarchy for simple families of languages extending the class of regular sets, we are led to the linear context-free languages and the deterministic context-free languages. However, the equivalence problem for the linear context free languages is undecidable, so that Theorem 1 rules out the existence of an efficient inference procedure. Similarly, construction of a recursively bounded inference system for the deterministic context-free languages would imply decidability of their equivalence problem, which is a long-standing open problem.

Theorem 2. Suppose that G is recursively enumerable, and that $L: G \twoheadrightarrow F$ is a recursive indexing. Then there is a system (ℓ,t) which conveys every grammar in G. If in addition the equivalence problem for L is decidable, then $S_{(\ell,t)}(G)$ is bounded above by an effectively computable function $R: G \to \mathbb{N}$.

Theorems 1 and 2 direct our search for easily inferable families of languages to the class of recursive indexings having a decidable equivalence problem. This leads us to ask whether this class enjoys any useful closure properties. For example, we observe that for any recursive indexings $L^0: \mathbb{N} \twoheadrightarrow F_0$ and $L^1: \mathbb{N} \twoheadrightarrow F_1$, we may recursively index $F_1 \cup F_2$ by "interleaving" L^0 and L^1; that is, define $L^2: \mathbb{N} \twoheadrightarrow F_0 \cup F_1$ by

$$L^2(2n) = L^0(n) \quad (\forall n \in \mathbb{N});$$
$$L^2(2n+1) = L^1(n) \quad (\forall n \in \mathbb{N}).$$

We would like to know whether decidability of the equivalence problems for L^0 and L^1 implies decidability for L^2. The following theorem indicates that this need not be the case.

Definition. Let F, F' be families of languages with $F \subseteq F'$, and let $L: G \twoheadrightarrow F$, $L': G' \twoheadrightarrow F'$ be recursive indexings. L is recursively embeddable in L' if there is an effectively computable map $e: G \to G'$ such that $(\forall G \in G)L(G) = L'(e(G))$.

Theorem 3. There exist recursive indexings $L^0: G^0 \twoheadrightarrow F^0$ and $L^1: G^1 \twoheadrightarrow F^1$ having decidable equivalence problems such that any L^2 into which both L^0 and L^1 are recursively embeddable must have an undecidable equivalence problem.

1.2.4. A Polynomial-Time Inference Algorithm for the Regular Languages

Notation. For simplicity, we restrict attention to the alphabet $\{0,1,2\}$. Let A be the set of all state graphs of complete, deterministic finite state (Moore) machines over this alphabet, R the class of regular subsets of $\{0,1,2\}^*$, and let $T: A \twoheadrightarrow R$ denote the map taking an automaton A to the set of tapes accepted by A. For $A \in A$, $\|A\|$ is to denote the number of states in the automaton A.

Definition. Let (ℓ,t) be a system which stably conveys every $A \in A$. (ℓ,t) is said to be polynomial bounded if there is a polynomial $p(x)$ such that $(\forall A \in A)\ S_{(\ell,t)}(A) \leq p(\|A\|)$.

Theorem 4. There is a set L of learners and a set T of teachers such that for any $\ell \in L$ and $t \in T$:

(i) (ℓ,t) stably conveys every $A \in A$;

(ii) the guess to which ℓ converges when presented A is the minimal automaton accepting $T(A)$;

(iii) (ℓ,t) is polynomial bounded.

Proof. Moore's minimization procedure for finite state machines 5, p. demonstrates an explicit construction for the minimal acceptor for a regular language L in terms of the equivalence classes of the relation:

$$x \equiv y \overset{\text{def}}{\longleftrightarrow} (\forall z \in\{0,1,2\}^*)\ xz \in L \leftrightarrow yz \in L$$

The learner of Theorem 4 will use this construction, given only partial knowledge about L.

In order to apply the construction, the learner ℓ must be able to do two things efficiently. First, representatives must be found for each of the equivalence classes. Second, by using these representatives together with certain other data about the language, the transitions between the equivalence classes must be established. Exactly which additional data are required will not be apparent to the learner. The teacher t (which, being equipped with the state graph, is in a position to determine which strings are of interest) will offer an appropriate set of strings to the learner. This set will depend on the representatives chosen by ℓ. t maintains a superset of the representatives by remembering the questions asked by ℓ.

Algorithm for learner ℓ

ℓ maintains a partially completed state graph of the minimal automaton for the language being presented, making incremental changes to the graph by adding new states and filling in transitions as the appropriate information is supplied by t.

Stage 0: Ask: "Is the empty string in L?". Make the empty string a representative.

Stage n ($n \geq 1$): Pick an undefined transition for some input symbol b from a state with representative x. (If no such transition exists, the learner knows that its construction is complete.)
Ask: "Is xb in L?" .

t will answer this question by returning a set of strings S_n, indicating for each y in S_n whether or not y is in L.
Call strings x and y inequivalent with respect to S_n if there is a string z such that xz, yz $\in S_n$ and xz $\in$ L $\leftrightarrow$ yz $\notin$ L. Otherwise, x and y are equivalent wrt S_n. There are 3 possible cases for ℓ:

(i) xb is not equivalent (wrt S_n) to any current representative r. ℓ makes xb a representative.

(ii) xb is equivalent to exactly one current representative r. ℓ adds the transition: $[x] \xrightarrow{b} [r]$ to its state graph.

(iii)xb is equivalent to more than one current representative. ℓ and t will be designed to prevent this case from occurring.

End of stage n.

End of learner algorithm.

Algorithm for teacher t

t is supplied with an n-state finite state machine M accepting language L. A set Q is maintained, consisting of all questions asked thus far by ℓ.

In response to the question "Is x in L?", t constructs a set of strings S, indicating to ℓ whether or not each string is in L.

1. Construction of S:

   Begin by placing x in S. For each q in Q such that $x \not\equiv q$, pick a string z such that $xz \in L \leftrightarrow qz \notin L$. Place xz and qz in S. (z may be chosen to be of length at most n-1).

2. Indicate to ℓ whether or not each string in S is in L.

End of Algorithm for t.

Correctness of algorithms: ℓ continues to add transitions to its state graph. When ℓ is attempting to define the b-transition from the state represented by x, the answers returned by t guarantee that either this arc points to the unique representative r which is equivalent to xb (case (ii) for ℓ); or that a new state [xb] is created (case (ii) for ℓ). It follows that after n+1 steps of dialog, ℓ has built a subset of the minimal automaton for L having n arcs. If the minimal automaton has k arcs, the first k+1 stages of dialog constitute stable conveyance of L, and in fact this can be accomplished in a number of steps polynomial in n.

--- x ---

The classes L and T alluded to in the statement of the theorem arise because of the wide variety of choices available to the learner, in picking representatives, and to the teacher, in picking interesting strings to present to the learner. One of the strengths of this theorem is that choosing <u>any</u> $\ell \in L$ and $t \in T$ yields a polynomial bounded system. Thus, learners and teachers succeed in their respective tasks by depending only on rather general assumptions about the strategy being

used by their dual, without knowledge of the specific algorithms involved.

The strategies sketched above depend crucially on close interaction between learner and teacher. In fact, a result due to Moore [5] indicates that such interaction is necessary.

Definition. An oracle is a teacher which always answers only the question asked of it.

Theorem 5 (Moore). For no oracle 0 and learner ℓ is the system $(\ell,0)$ polynomial bounded.

Theorem 5, when viewed in light of Theorem 4, lends a good deal of insight into the situation faced by the learner. Namely, there is a relatively small number of strings from a language which is sufficient to allow that language to be identified efficiently. The problem lies in finding those strings.

We remark in passing that applying analogous techniques to the family of recognizable tree languages [6] yields an efficient system for conveying this family. This is important because the recognizable tree languages possess much of the structural richness of the context-free languages.

1.2.5. The R^{k}_{C} Hierarchy

We now consider an infinite hierarchy of language families $R = R^{0}_{C} \subsetneq R^{1}_{C} \subsetneq R^{2}_{C} \subsetneq \cdots$. These languages are of interest because the technique outlined in the proceeding section may be modified to yield an efficient system to convey all the languages at any particular level of the hierarchy.

Definitions

(i) Let $x = a_1a_2\cdots a_n c b_n b_{n-1}\cdots b_1$ be a string, where $n \geq 0$; $a_i, b_i \in \Sigma$ $(\forall 1 \leq i \leq n)$; and $c \in \Sigma \cup \{\Lambda\}$. Then $x^{\supset}$ (read x folded) is by definition the string $a_1b_1a_2b_2\cdots a_nb_nc$.

(ii) For $x \in \Sigma^*$, $x^{\subset}$ (x unfolded) is the unique $y \in \Sigma^*$ such that $y^{\supset} = x$.

(iii) For $k \geq 0$ define $\overset{k}{\subset}$ and $\overset{k}{\supset}$ by induction on k via:

$$x^{\overset{0}{\subset}} = x ; \quad x^{\overset{k+1}{\subset}} = (x^{\overset{k}{\subset}})^{\subset} ;$$
$$x^{\overset{0}{\supset}} = x ; \quad x^{\overset{k+1}{\supset}} = (x^{\overset{k}{\supset}})^{\supset} .$$

(iv) If L is a language, define $L^{\overset{k}{\subset}} = \{x^{\overset{k}{\subset}} \mid x \in L\}$. For F a family of languages, define $F^{\overset{k}{\subset}} = \{L^{\overset{k}{\subset}} \mid L \in F\}$. $L^{\overset{k}{\supset}}$ and $F^{\overset{k}{\supset}}$ are defined similarly.

Proposition

(i) If F is a family of languages, $F^{\overset{k}{\subset}} = \{L \mid L^{\overset{k}{\supset}} \in F\}$.

(ii) If L is regular, $L^{\supset}$ is also. Hence, $R \supseteq R^{\supset}$.

(iii) $R^{\overset{k}{\subset}} \subseteq R^{\overset{k+1}{\subset}}$.

(iv) $\{ww^R \mid w \in \{0,1\}^*\}^{\overset{k}{\subset}} \in R^{\overset{k+1}{\subset}} - R^{\overset{k}{\subset}}$; hence $(\forall k \in \mathbb{N})\ R^{\overset{k}{\subset}} \subsetneq R^{\overset{k+1}{\subset}}$.

Proof. (i) follows from the fact that $\subset$ and $\supset$ are inverse operations. (ii) can be proved by using a finite automaton for L to construct one for $L^{\supset}$. (iii) then follows by induction on k, using (i) and (ii). (iv) is clear from the preceding, thus establishing that the $R^{\overset{k}{\subset}}$'s form a strictly increasing hierarchy.

$R^{\overset{k}{\subset}}$ may be indexed by $T_k: \Lambda \twoheadrightarrow R^{\overset{k}{\subset}}$, where $T_k(A) = T(A)^{\overset{k}{\subset}}$ $(\forall A \in \Lambda)$. Theorem 4 then generalizes to arbitrary levels of the $R^{\overset{k}{\subset}}$ hierarchy.

Theorem 6. There is a set L of learners and a set T of teachers such that for any $\ell \in L$ and $t \in T$:

(i) (ℓ,t) stably conveys every $A \in \mathcal{A}$ with respect to the indexing T_k.

(ii) The guess to which ℓ converges when presented A is the minimal automaton A' such that $T_k(A') = T_k(A)$.

(iii) There is a polynomial $p(x)$ such that $(\forall A \in \mathcal{A})\ S_{(\ell,t)}(A) \leq p(\|A\|)$.

Proof. The idea behind the proof is that any (ℓ,t) which stably conveys every $A \in \mathcal{A}$ with respect to the indexing $T: A \twoheadrightarrow R$ may be modified, without much loss of efficiency, to yield a system (ℓ',t') which stably conveys every $A \in \mathcal{A}$ with respect to T_k. The modification consists of k-folding and k-unfolding questions and answers at strategic points.

There is a simple grammatical characterization of $R^{\overset{1}{C}}$.

Definition

(i) A context-free grammar $G = \langle V,\{0,1,2\},P,S\rangle$ is said to be restricted even linear$^{(2)}$ if it satisfies the following conditions:

a) All rules are of the form

$$\left.\begin{array}{l} A \to aBb \\ A \to a \\ A \to \Lambda \end{array}\right\} \quad \begin{array}{l} \text{where } a, b \in \{0,1,2\}; \\ A, B \in V - \{0,1,2\} \end{array}$$

b) For no $a, b \in \{0,1,2\}$ and $A, B, C \in V - \{0,1,2\}$ such that $B \neq C$ it is the case that $A \to aBb$ and $A \to aCb$ are both in P.

(ii) Let E denote the class of restricted even linear grammars.

(iii) EL will denote the class of languages generated by grammars in E.

(iv) Define a recursive indexing $E: E \twoheadrightarrow EL$ by mapping each $G \in E$ to the language generated by G.

Proposition. $EL = R^{\overset{1}{C}}$.

Theorem 7. There is a system (ℓ,t) which stably conveys every $G \in E$ with respect to the indexing E. Moreover, $S_{(\ell,t)}(G)$ is bounded above by a polynomial in the number of non-terminals in G.

Finally, we indicate the relationship between the $R^{\overset{k}{C}}$'s and some of the standard classes of languages.

Definition. $R^{\overset{\infty}{C}} = \bigcup_{k \in \mathbb{N}} R^{\overset{k}{C}}$.

Theorem 8. $\{0^n 1^n \mid n \geq 1\} \circ \{2\}^* \notin R^{\overset{\infty}{C}}$.

Corollary

(i) $R^{\overset{\infty}{C}} \supset$ linear context free

(ii) $R^{\overset{\infty}{C}} \supset$ deterministic context free

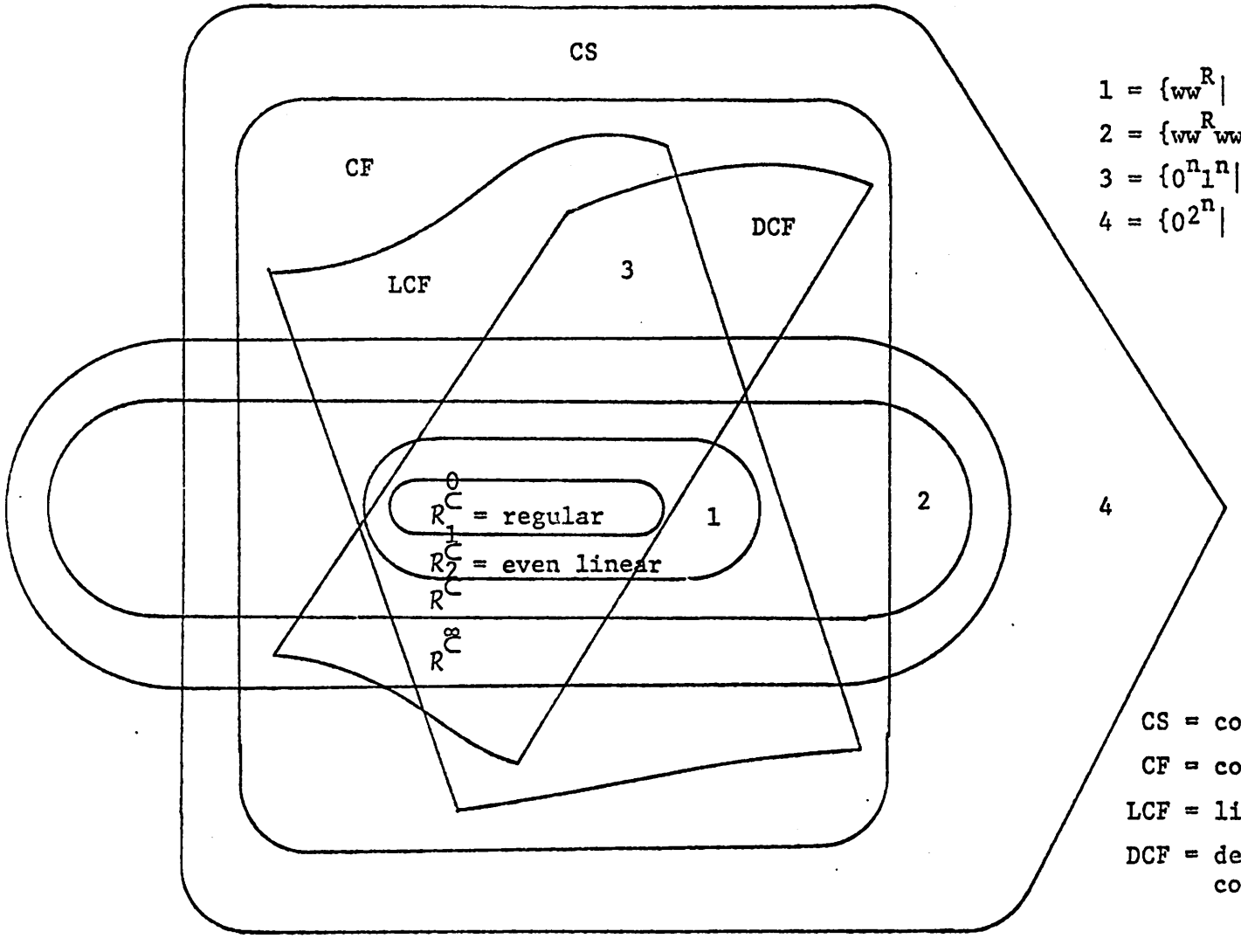
Proposition

(i) $\{ww^R \mid w \in \{0,1\}^*\} \in R^{\overset{1}{C}}$ - (deterministic context free)

(ii) $\{(ww^R)^2 \mid w \in \{0,1\}^*\} \in R^{\overset{2}{C}}$ - (context free)

(iii) $\{0^{2^n} \mid n \geq 1\} \in$ (context sensitive) $- R^{\overset{\infty}{C}}$.

These relationship appear in the following diagram.



1 = $\{ww^R \mid w \in \{0,1\}^*\}$

2 = $\{ww^Rww^R \mid w \in \{0,1\}^*\}$

3 = $\{0^n1^n \mid n \geq 1\} \cdot \{2\}^*$

4 = $\{0^{2^n} \mid n \geq 1\}$

CS = context sensitive

CF = context free

LCF = linear context free

DCF = deterministic context free

Footnotes

(1) As our results deal only with the existence of polynomial and recursive time bounds, the exact encoding chosen is not important.

(2) Even linear languages have been studied by Amar and Putzol [1]. Our notion of restricted even linear grammars constitutes a normal form for general even linear grammars.

References

[1] V. Amar and G. Putzolu, "On a family of linear grammars," Information and Control 7 (1964), 283-291.

[2] L. Blum and M. Blum, "Inductive inference: a recursion-theoretic approach," Memorandum ERL-M386, Electronics Research Laboratory, University of California, Berkeley, March 1973.

[3] E. Mark Gold, "Language identification in the limit," Information and Control 10 (1967), 447-474.

[4] J. Horning, "A study of grammatical inference," Technical Report CS 139, Computer Science Department, Stanford University, 1969.

[5] E.F. Moore, "Gedanken experiments on sequential machines," in Automata Studies, Princeton University Press, 1956.

[6] J.W. Thatcher, "Characterizing derivation trees of context-free grammars through a generalization of finite automata theory," J.C.S.S. 1 (1967), 317-322.

[7] R.M. Wharton, "Grammatical inference and approximation," Technical Report 51, Department of Computer Science, University of Toronto, 1973.

§2. FINITE AUTOMATA IN 2- AND 3-DIMENSIONAL SPACE*

Introduction

This chapter 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 [1] 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 [4] 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 chapter discusses work done jointly by Manuel Blum and William Sakoda.

An old but still very strong mathematical argument due to M. Minsky [6] 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 2.1, 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. 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 universal on a 1-dimensional tape, no matter how that tape is marked.

In Section 2.1, 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

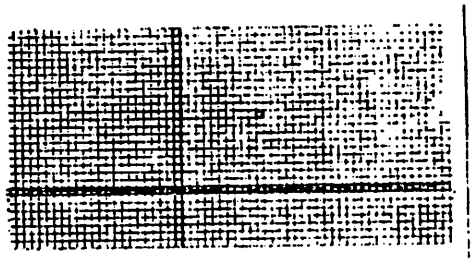


Figure 2.1

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 2.2 suggests one simple algorithm. In fact, 2 finite automata and 1 pebble can simulate a universal 2-counter automaton (Sipser [8]). 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 showed that a finite automaton with a finite collection of pebbles cannot entirely search (thread) an arbitrary finite graph. S. Cook [5] and C. Rackhoff [7]

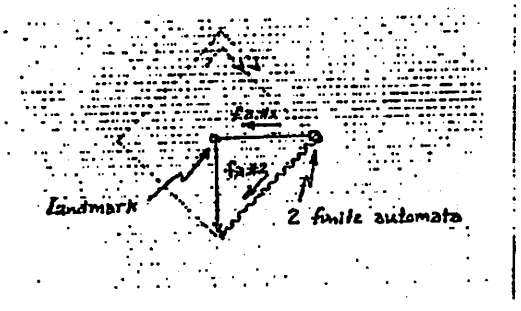


Figure 2.2

have since obtained upper and lower bounds on the number of pebbles needed to search a graph with n nodes. In Section 2.2 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.

2.1 Searching 2-Dimensional Space

2.1.1 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 2.3. 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, Z, to encode the contents of 2 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

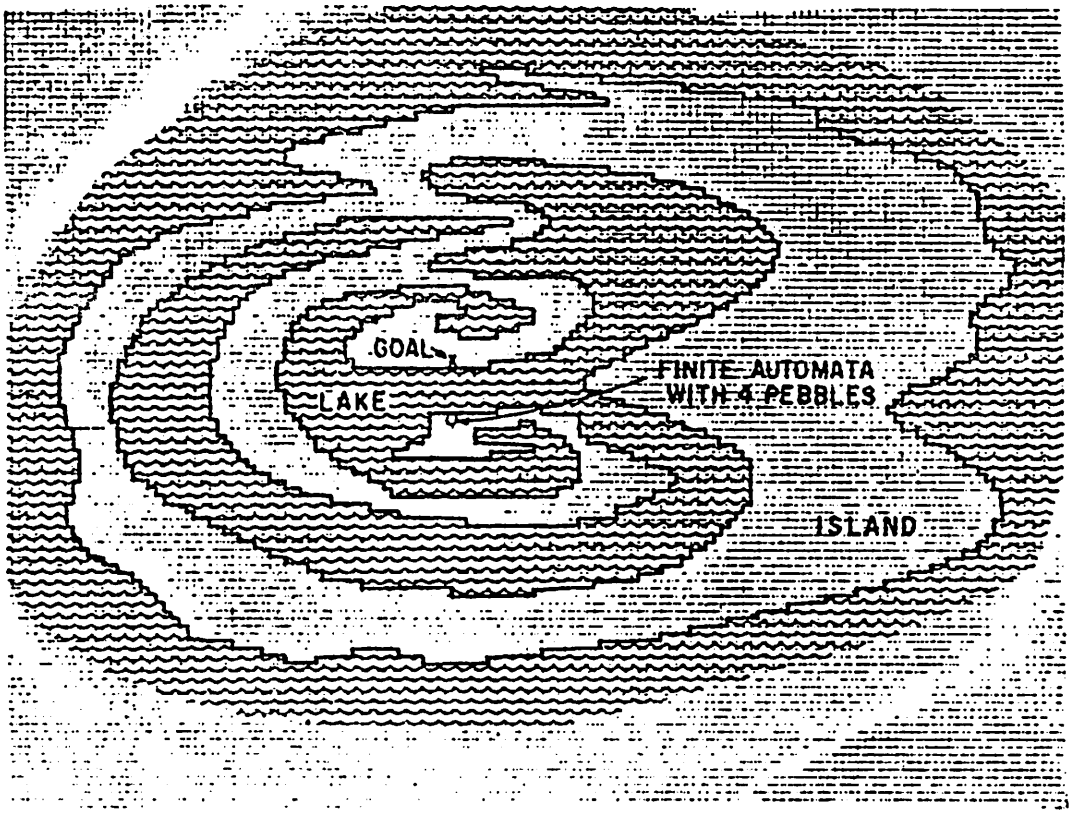


Figure 2.3

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 without 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 searched, enlarged, then searched again (Figure 2.4). 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.

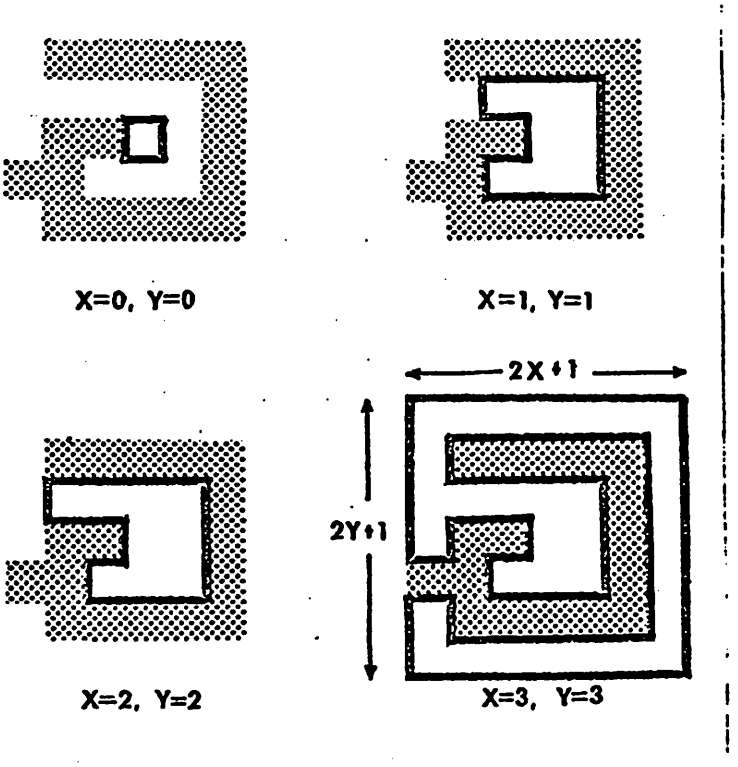


Figure 2.4

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 0 and a "count" pebble C. A squadron of 4 pebbles under the control of 1 finite automaton moves between the 2 pebbles 0 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 in detail. 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 pebbles 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 an obstructed 2-dimensional space.

2.1.2 Counter Construction

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.

Problem: 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 0 in a mission accomplished state, it may be restarted in an increment or decrement state, and this constitutes the beginning of stage n+1. At the end of a stage, C is to occupy the same position as 0 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 0 to C, moves C appropriately, and then shuttles back to 0. 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 0 to C.

Algorithm: The shuttle is to follow the instructions below until C or 0 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. Figure 2.5). From this other side of the lake, the shuttle is to continued its movement north following the directions given above.

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. Figure 2.6). 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. Figure 2.7). 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 <u>between-stages position</u> of C. Shown in Figure 2.8 are the counter contents or numbers represented by a succession of between-stages positions.

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

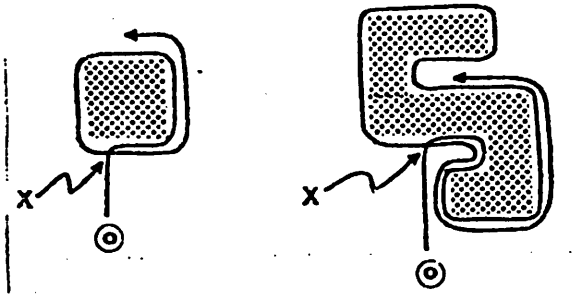


Figure 2.5

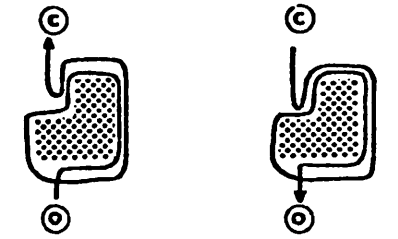


Figure 2.6

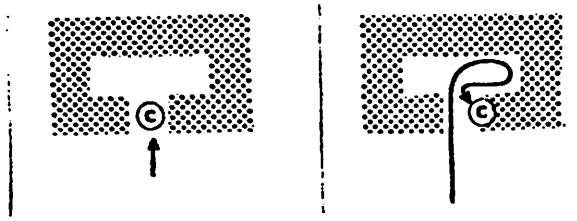


Figure 2.7

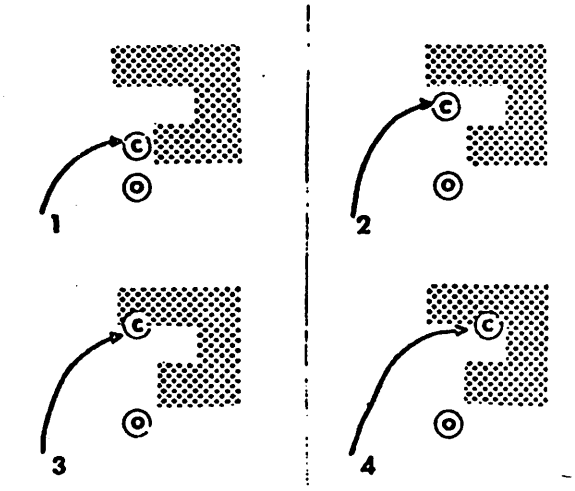


Figure 2.8

positions of C are unique. For example, the successive positions of the shuttle as it moves from O to C are compared in Figure 2.9 below with the successive between-stages positions of C as it is incremented stage after stage. Note the jump in Figure 2.9 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 2.10.

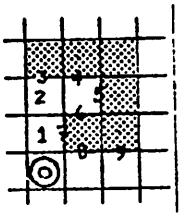
Of course, the shuttle does not have wings. It cannot move the count pebble instantly from position 56 in Figure 2.10 to position 57. It does this gradually instead.

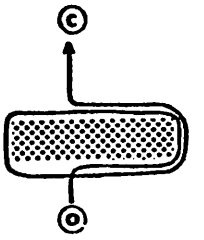
The procedure used by the shuttle to <u>increment</u> the count pebble is as follows:

<u>Algorithm</u>: From O the shuttle moves toward C (north, counter-clockwise around water, etc.) until it "encounters" C. We say that the shuttle encounters C 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 2.11 shows C inside a cell or on one of its 2 edges, and the movement of the shuttle until it encounters C.

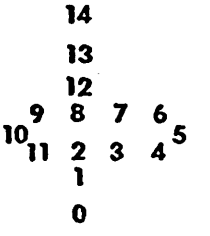
After encountering C, the shuttle determines which of the 3 cases below holds, and moves C accordingly:

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 (Figure 2.12).



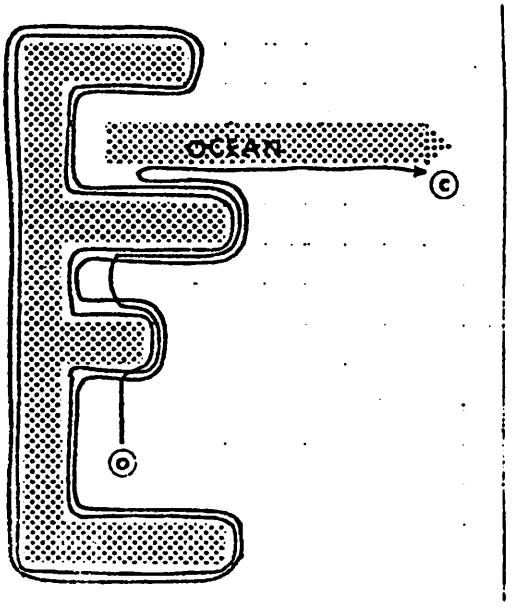


shuttle movement



count pebble positions

Figure 2.9



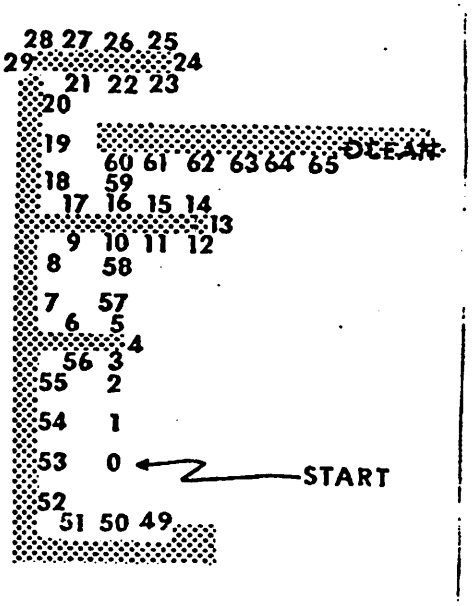


Figure 2.10

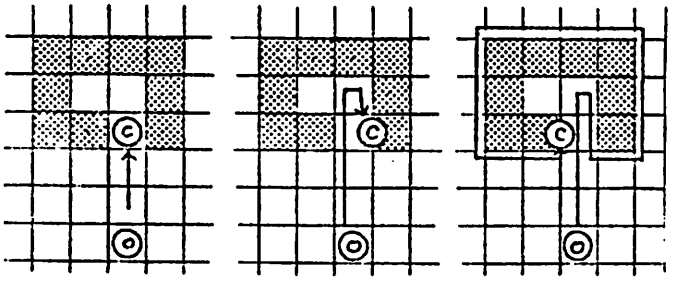


Figure 2.11

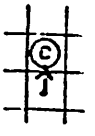


Figure 2.12

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 0). See Figure 2.13.

3. If C lies on a lakeshore edge as in Figure 2.14, 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 0.)

After moving C the shuttle wends its way back to 0 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 0 to C or back). This position must be in the column containing 0. The shuttle now checks if there is accessible land due north of C. If not, it returns to 0 in the "mission impossible" state. If yes, it moves C to the interior of the nearest land cell due north of its present position (Figure 2.15).

Now the shuttle continues its way back to 0. (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.)

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.

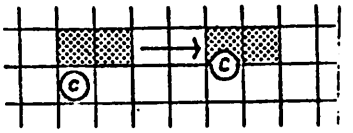


Figure 2.13

Figure 2.14

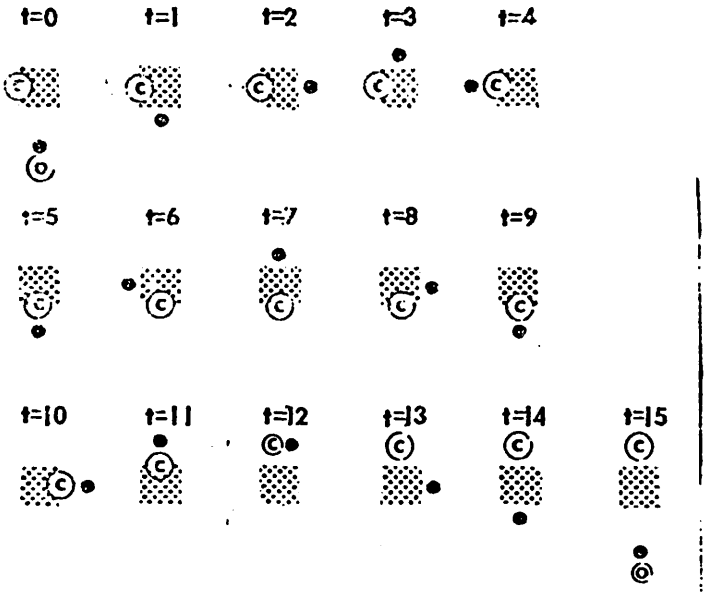


Figure 2.15

Algorithm. 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: Move C to the interior of that cell. (See Figure 2.16a.)

Case 3: C is on an edge and is reached from an edge. Move C to the edge that led to it. (See Figure 2.16b.)

Case 4: C is in the interior of a cell and the southern edge of that cell is on a shore (Figure 2.17). 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 O. 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.

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

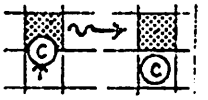


Figure 2.16a

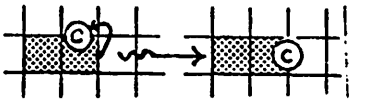


Figure 2.16b

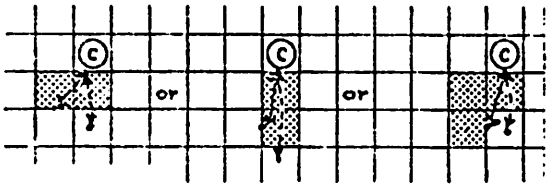


Figure 2.17

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 0. 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 0.

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 0. (In moving from C to 0, 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 0. 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 0). 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 counterclockwise 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.)

2.2 3-Dimensional Space is Unsearchable

In Section 2.1, we showed that a finite state machine with 7 pebbles 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.

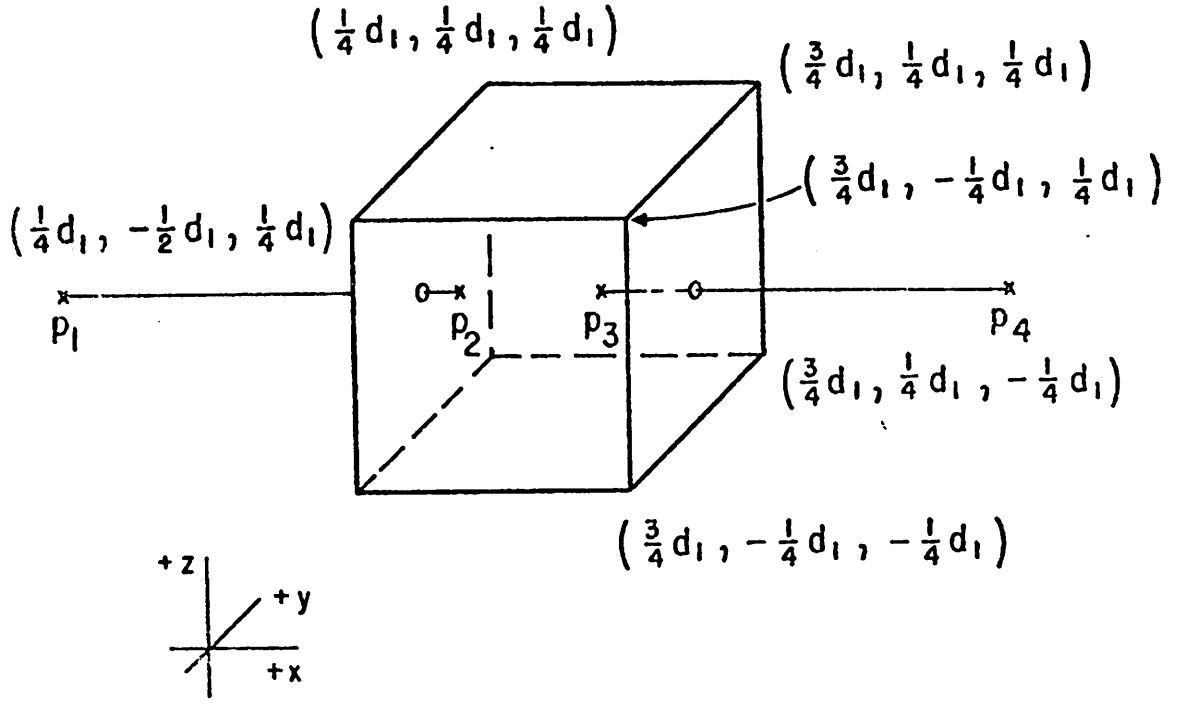
Let $U = \{(0,0,1),(0,0,-1),(0,1,0),(0,-1,0),(1,0,0),(-1,0,0)\}$.
Up Down N S E W

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 $\{v \in Z^3 \mid (\text{there exists } u \in U)\ \{v,v+u\} \in G\}$. 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) (Figure 2.18), 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.



$$d_i = 8s^2\ell^2 + 4\ell$$

$$\ell = \text{lcm}\left\{2, \ldots, s\right\}$$

$$p_1 = (0, 0, 0)$$
$$p_2 = (\tfrac{1}{4}d_1 + 1, 0, 0)$$
$$p_3 = (\tfrac{3}{4}d_1 - 1, 0, 0)$$
$$p_4 = (d_1, 0, 0)$$

Figure 2.18

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.

(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.

(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 2.18.

Definition. For $x, y \in Z$, let $\langle x,y \rangle$ denote the point $(\ell \times x, \ell \times y, 0)$ where ℓ will henceforth denote $\text{lcm}\{2,\ldots,s\}$. The $\langle x,y \rangle$ are the connector points. Let $C = \{\langle x,y \rangle | x,y \in Z\}$.

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 $a \in N$ and $u \in U$. For such v_1 and v_2, each arc in the set $\{\{v_1+bu, v_1+(b+1)u\} \mid 0 \leq b < a\}$ is between v_1 and v_2.

Definition. (i) Let $c_1, c_2, \ldots, c_9$ be defined as in Figure 2.19. $C_x((0,0,0))$, 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 $1 \leq i \leq 8$.

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

Definition. (i) Let $d_1, d_2, \ldots, d_9$ be defined as in Figure 2.20. $C_y((0,0,0))$, the y-connector at (0,0,0), consists of all arcs lying between a pair of points d_i, d_{i+1} for some $1 \leq i \leq d$.

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

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

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 $\langle x_1, y_1 \rangle$ and $\langle x_2, y_2 \rangle$, 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.

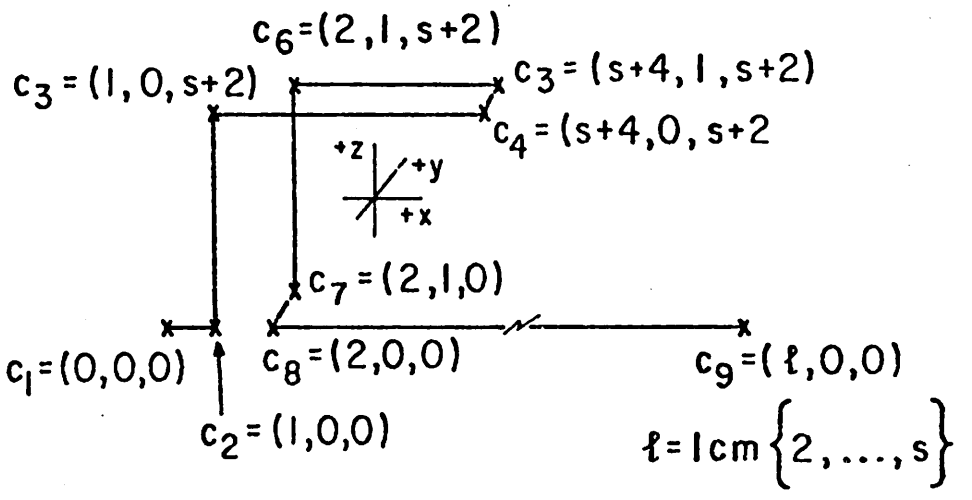

$c_6=(2,1,s+2)$
$c_3=(1,0,s+2)$
$c_3=(s+4,1,s+2)$
$c_4=(s+4,0,s+2$
+z
+y
+x
$c_7=(2,1,0)$
$c_1=(0,0,0)$
$c_8=(2,0,0)$
$c_9=(\ell,0,0)$
$c_2=(1,0,0)$
$\ell=\text{lcm}\{2,\ldots,s\}$

Figure 2.19

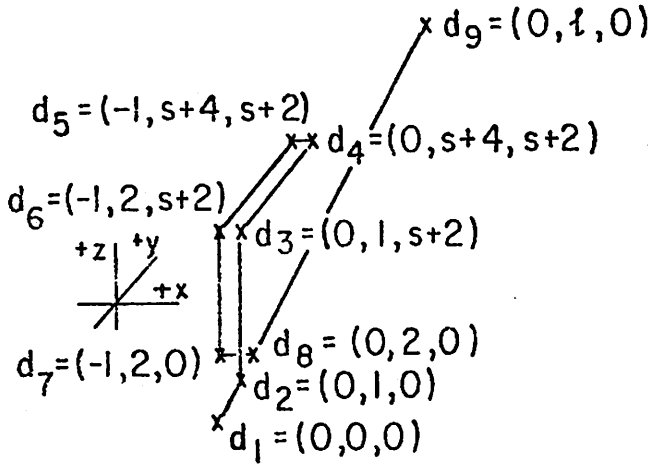

d9=(0,1,0)
d5=(-1,s+4,s+2)
d4=(0,s+4,s+2)
d6=(-1,2,s+2)
d3=(0,1,s+2)
+z
+y
+x
d7=(-1,2,0)
d8=(0,2,0)
d2=(0,1,0)
d1=(0,0,0)

Figure 2.20

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

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 $x, y \in Z$, $G = \{\{v+\langle x,y\rangle, v'+\langle x,y\rangle\} \mid \{v,v'\}\in G\}$. (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, $\langle x_i,y_i\rangle$, 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, $j \leq s+1$. For any $k \geq 1$ the points $\langle x_k,y_k\rangle$ and $\langle x_{k+1},y_{k+1}\rangle$ 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. $h = 4s^2$; $w = h \times \ell$.

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

Definition. (i) Let $f_1, \ldots, f_9$ be as in Figure 2.22. 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 $p \in Z^3$, $B_y(h,p) = \{\{v+p, v'+p\} | \{v,v'\} \in B_y(h,(0,0,0))\}$.

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

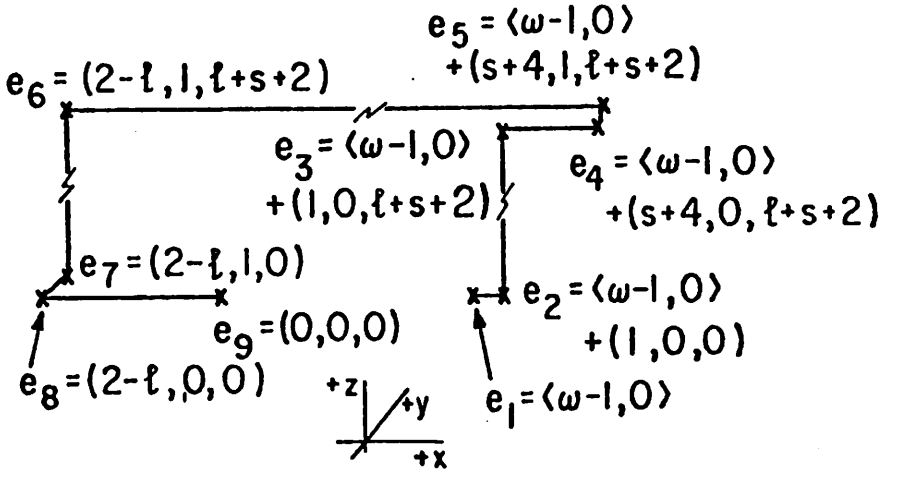

e5 = ⟨ω-1,0⟩
+(s+4,1,ℓ+s+2)
e6 = (2-ℓ,1,ℓ+s+2)
e3 = ⟨ω-1,0⟩
+(1,0,ℓ+s+2)
e4 = ⟨ω-1,0⟩
+(s+4,0,ℓ+s+2)
e7 = (2-ℓ,1,0)
e2 = ⟨ω-1,0⟩
+(1,0,0)
e9 = (0,0,0)
e8 = (2-ℓ,0,0)
e1 = ⟨ω-1,0⟩
+z
+y
+x

Figure 2.21

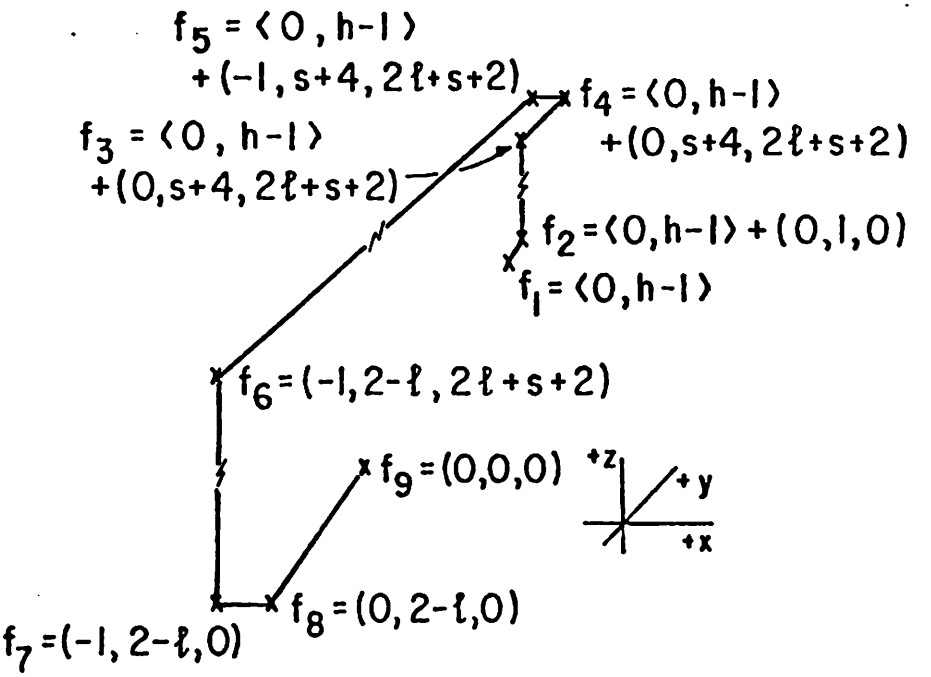

f5 = ⟨0, h-1⟩
+(-1, s+4, 2ℓ+s+2)
f4 = ⟨0, h-1⟩
+(0, s+4, 2ℓ+s+2)
f3 = ⟨0, h-1⟩
+(0, s+4, 2ℓ+s+2)
f2 = ⟨0, h-1⟩ + (0, 1, 0)
f1 = ⟨0, h-1⟩
f6 = (-1, 2-ℓ, 2ℓ+s+2)
f9 = (0, 0, 0)
+z
+y
+x
f8 = (0, 2-ℓ, 0)
f7 = (-1, 2-ℓ, 0)

Figure 2.22

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

(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 other 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 $1*(w+2) = 4s^2l^2+2l$. Lead wires into this cube as in Figure 2.18. 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 $\langle 0,0\rangle$, 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 $\langle 0,0\rangle$ 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 together 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 then 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 $x, y \in Z$, $f(\langle x,y\rangle) = \langle [x]_w, [y]_h \rangle$.

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 $p_1, p_2, \ldots, p_{r-1}$ 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 $2 \leq i \leq r$.

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 $1 \leq i \leq 8$,

$$T(\langle a,b\rangle + c_i) = f(\langle a,b\rangle) + e_i$$

(c_i as specified in Figure 2.19).

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

$$T(\langle a,b\rangle + c_i) = f(\langle a,b\rangle) + c_i$$

(cf. Figure 2.21).

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

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

(cf. Figure 2.20).

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

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

(cf. Figure 2.22).

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

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 2.19 and 2.21. 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 $1 \leq i \leq s+2$, 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 $1 \leq i < j \leq s+1$ such that $p_j = p_i$. The computation then repeats according to the pattern $p_b = p_{[b-i]_{j-i}+i}$ for $j \leq b \leq s+2$.

For $1 \leq r \leq \ell+s+2$ 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:

$$\hat{p}_r = p_r \qquad \text{for } 1 \leq r < i ;$$
$$\hat{p}_r = p_{[r-i]_{j-i}+i} \qquad \text{for } 1 \leq r \leq \ell+s+2 .$$

In particular,

$$\hat{q}_{k+\ell} = \hat{p}_{\ell+s+2} = p_{[\ell+s+2-i]_{j-i}+i}$$
$$= p_{[s+2-i]_{j-i}+i} \quad (\text{since } (j-i) \mid \ell = \text{lcm}\{2,\ldots,s\})$$
$$= q_{k+1} . \qquad \text{Q.E.D.}$$

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

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

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

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

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

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

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 $R'(p_1) = \{f(v) | v \in R(p_1)\}$.

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' $(2 \leq i \leq r)$, there is a $v_i \in R(p_1)$ such that $d(p_i', v_i) \leq s-1$. By (ii) of the preceding lemma, $d'(f(p_i'), f(v_i)) \leq d(p_i', v_i)$. By the Projection Theorem $p_i = f(p_i')$, so $d'(p_i, f(v_i)) \leq s-1$ 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 $(2 \leq i \leq r)$ there is $v_i \in R'(p_2)$ such that $d'(p_i, v_i) \leq s-1$. By definition of R', $v_i + (p_1 - p_2) \in R'(p_1)$. Then $d'(p_i, v_i + (p_1 - p_2)) \leq d'(p_i, v_i) + d'(v_i, v_i + (p_1, p_2)) = d'(p_i, v_i) + 1 \leq s$.

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

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

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

(iii) $H(y) \cup V(x) \cup R_2'(\langle x,y\rangle)$, where

$$H(y) = \{\langle a,y\rangle \mid 0 \leq a < w\} ;$$

$$V(x) = \{\langle x,b\rangle \mid 0 \leq b < h\} ;$$

$$R_2'(\langle x,y\rangle) = \{\langle [na+x]_w, [nb+y]_h\rangle \mid n,a \in \mathbb{N}^+,\ b \in Z-\{0\},\ a+|b| \leq s\} .$$

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

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

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 $\langle [\hat{n}\hat{a}+x]_w, [\hat{n}\hat{b}+y]_h\rangle$ of set (ii) where $\hat{a}$ is required to be non-negative. The values $\hat{n} = n(wh-1)$, $\hat{a} = -a$, $\hat{b} = -b$ are appropriate:

$$\begin{aligned}\langle [na+x]_w, [nb+y]_h\rangle &= \langle [na+x]_w, [nb+y]_h\rangle + \langle [nwh(-a)]_w, [nwh(-b)]_h\rangle \\ &\quad \text{(since the second term is } \langle 0,0\rangle) \\ &= \langle [n(wh-1)(-a)+x]_w, [n(wh-1)(-b)+y]_h\rangle \\ &= \langle [\hat{n}\hat{a}+x]_w, [\hat{n}\hat{b}+y]_h\rangle .\end{aligned}$$

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

(iv) equals (iii): To show: $R_2'(\langle x,y\rangle) \subseteq F(\langle x,y\rangle)$, the other containment being easy. Let $\langle [na+x]_w, [nb+y]_h\rangle \in R_2'(\langle x,y\rangle)$. Pick $q \in \mathbb{N}^+$ such that $na = q*w + [na]_w$. Since $0 < a \leq s$, $a|w = h*lcm\{2,3,\ldots,s\}$, and this implies $a|[na]_w$. Therefore $na = q*w + \hat{n}a$ with $\hat{n} \in \mathbb{N}$ and $\hat{n}a < w$. Then

$$<[na+x]_w,[nb+y]_h> = <[q*w+\hat{n}a+x]_w,[((q*w+\hat{n}a)/a)b+y]_h>$$
$$= <[\hat{n}a+x]_w,[\hat{n}b+y]_h>$$

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

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

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 $m_2 = <s+1,2s^2>$ is not within distance s of $R'(<0,0>)$.

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

$$R'(<0,0>) = H(0) \cup V(0) \cup F(<0,0>)$$

where $F(<0,0>) = \{<na,[nb]_h> | n,a \in \mathbb{N}^+,\ b \in Z-\{0\},\ 0 \leq na < w,\ a+|b| \leq s\}$. 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 $p \in F(<0,0>)$. Suppose, to the contrary, that there is $<na,[nb]_h> \in F(<0,0>)$ such that

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

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

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

$$\begin{aligned} d'_y(2s^2,[nb]_h) &= \min([2s^2-[nb]_h]_h,[[nb]_h-2s^2]_h) \\ &\quad \text{(where } h = 4s^2\text{)} \\ &\geq \min[(2s^2-(2s^2-s-1)]_h,[(2s^2-s-1)-2s^2]_h) \\ &= [2s^2-[2s^2-s-1)]_h \\ &= s+1 \end{aligned}$$

(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

$$\begin{aligned} d'_y(2s^2,[nb]_h) &= \min([2s^2-[nb]_h]_h,[[nb]_h-2s^2]_h) \\ &\geq [-4s^2+s+1]_h = s+1 \end{aligned}$$

(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 $\langle 0,0\rangle$ is not within distance s of $R'(m_2)$.

Sublemma 2 follows from Sublemma 1, together with:

Symmetry Lemma. Let $0 \le x_1, x_2 < w$ and $0 \le y_1, y_2 < h$. 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 $n \in \mathbb{N}$, $a,b \in Z$ with $|a| + |b| \le s$ such that $p_2 = \langle [na+x_2]_w, [nb+y_2]_h\rangle$. By hypothesis,

$$
\begin{aligned}
d &= d'(\langle x_1, y_1\rangle, \langle [na+x_2]_w, [nb+y_2]_h\rangle) \\
&= \min([x_1 - [na+x_2]_w]_w, [[na+x_2]_w - x_1]_w) \\
&\quad + \min([y_1 - [nb+y_2]_h]_h, [[nb+y_2]_h - y_1]_h) \\
&\quad \text{(definition of } d') \\
&= \min([[x_1 + n(-a)]_w - x_2]_w, [x_2 - [x_1 + n(-a)]_w]_w) \\
&\quad + \min([[y_1 + n(-b)]_h - y_2]_h, [y_2 - [y_1 + n(-b)]_h]_h) \\
&= d'(p_1, \langle x_2, y_2\rangle)
\end{aligned}
$$

where $p_1 = \langle [x_1+n(-a)]_w, [y_1+n(-b)]_h\rangle$. Since $|(-a)| + |(-b)| = |a| + |b| \le s$, $p_1 \in R'(\langle x_1, y_1\rangle)$ 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

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

[2] M. Blum and C. Hewitt, "Automata on a 2-Dimensional Tape," IEEE Conference Record, 8th Annual Symposium on Switching and Automata Theory, 1967, 155-160.

[3] L. Budach, "On the Solution of the Labyrinth Problem for Finite Automata," Elektronische Informationverarbeitung und Kybernetik EIK 11 (1975), 10-12, 661-672.

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

[5] S. Cook, in preparation.

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

[7] C. Rackhoff, in preparation.

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