LOGIC OF COMPUTERS GROUP

Department of Computer and Communication Sciences
2080 Frieze Building
The University of Michigan
Ann Arbor, Michigan 48104
ERGODIC MACHINES--PROBABILISTIC AND APPROXIMATE HOMOMORPHIC SIMPLIFICATIONS

Sudhir Aggarwal

August 1975

Technical Report No. 184

with assistance from:

National Science Foundation
Grant No. DCR71-01997
Washington, D.C.
CHAPTER I

INTRODUCTION

Since the age of the electronic computer, modelling and simulation has become an extremely important and popular activity in the investigation of complex problems. We find computer simulation models being used in every imaginable discipline. These simulation models usually have the common feature of being simplifications of a more complex, hypothesized "base" model. In the study of the brain or the economy, for example, we do not have the capability of simulating the base model that we imagine represents reality, and instead, we are forced to simulate a more tractable "lumped" model. The models that we simplify in order to simulate often appear conceptually as probabilistic models. This thesis investigates methods to study probabilistic models as well as the simplification of probabilistic models. Before elaborating on this, we discuss the formal framework that we shall use and we make clear what we mean by a model.

1.1 The Formal Framework

Recent work by Zeigler [41-45] has provided a formal mathematical foundation for modelling and simulation. In this approach, modelling and simulation is viewed as the study of the interrelationship among the quintuple of Real World, Base Model, Lumped Model, Computer, and Experimental Frame. The Real World is considered to be a set of input-output data. The modeller conceptualizes his structural beliefs
about the real world by constructing a base model that is capable of
counting for the real world input-output data. This base model is in
general very complex and cannot be studied by simulations on a computer.
Thus, of necessity, the modeller constructs a lumped model that can be
simulated and that accounts for some input-output behavior of interest.
Since the lumped model is a simplification, we do not expect it to be
able to explain all the input-output behavior. The role of the experi-
mental frame is to provide a formal method to restrict the input-output
behavior that a lumped model must match. Different experimental franes
of course yield different possible valid lumped models. Finally, the
conceptual lumped model, which is considered as simply a mathematical
entity, must be translated into a program that can be run on the com-
puter. If we let R be the Real World, B the Base Model, L the lumped
Model, and C the Computer, we can visualize the process as one involving
the following transformations: $R \rightarrow B \rightarrow L \rightarrow C$. The transformation $B \rightarrow L$
is the simplification stage, and it is here that model validation should
also be found. We shall be studying this stage of the transformation
in the case of a probabilistic base model. We study simplifications to
both probabilistic as well as deterministic lumped models. The stage
$L \rightarrow C$ is a verification stage that checks that the computer model in
fact does what it is supposed to.

This approach may be compared with a more typical approach. In
the more classical approach, one considers essentially $R \rightarrow L \rightarrow C$. The
validation stage now involves comparing $R$ to $C$. Unfortunately, since $R$
is the real system and $C$ the computer model, the criteria for validation
appear imprecise and ad hoc, often depending totally on statistical
tests.
ERGODIC MACHINES--PROBABILISTIC AND APPROXIMATE

HOMOMORPHIC SIMPLIFICATIONS

by

Sudhir Aggarwal

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Computer and Communication Sciences) in The University of Michigan

1975

Doctoral Committee:

Associate Professor Bernard P. Zeigler, Chairman
Professor Frederick J. Beutler
Professor Arthur W. Burks
Professor John H. Holland
Associate Professor John F. Meyer
ACKNOWLEDGEMENTS

I express my appreciation to Bernard Zeigler, Frederick Beutler, Arthur Burks, John Holland, and John Meyer for their faithful service as members of my committee. John Holland first sparked my interest in this area of computer science, and Arthur Burks first initiated me into research. Most of all, I thank Bernard Zeigler for giving so generously of his time, his ideas and his experience. He was always ready to discuss any problem, and invariably successful in suggesting new leads.

The Logic of Computers Group provided me with partial financial support as well as a congenial atmosphere in which to pursue research. I am grateful to the present and past members of this group for their many kindnesses. In particular, a sincere thank you to Albert Bethke with whom I spent long hours developing and testing the simulation package.

The moral support that I needed was derived from my wife, Mridula. My parents and my brother provided encouragement.

The excellent and speedy job of typing was done by Alice Gantt to whom I extend my thanks and my appreciation.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS .................................................. ii  
LIST OF TABLES ...................................................... v  
LIST OF ILLUSTRATIONS .............................................. vi  
LIST OF APPENDICES ................................................... viii

Chapter

I. INTRODUCTION ...................................................... 1  
   1.1 The Formal Framework ......................................... 1  
   1.2 Homomorphisms ............................................... 3  
   1.3 The Context .................................................. 4  
   1.4 Thesis Organization ......................................... 6

II. ERGODIC MACHINES ................................................ 8  
   2.1 Ergodic Transformations ...................................... 8  
   2.2 The Ergodic Theorem ......................................... 10  
   2.3 An Ergodic Machine .......................................... 11  
   2.4 Probabilistic Automata ....................................... 12  
   2.5 Ergodic Machines and Probabilistic Automata ............... 15  
   2.6 Information .................................................. 22  
   2.7 Canonical Ergodic Machines and Multiplicative Generators ............................................... 32  
   2.8 Universally Faithful Transformations ......................... 38  
   2.9 Pseudo-Random Number Generators ......................... 54  
   2.10 Parallel Seeds and Permutation Independence .......... 56  
   2.11 Nonautonomous Ergodic Machines ............................ 64  
   2.12 Conclusion .................................................. 65

III. APPROXIMATE HOMOMORPHISMS--SIMPLIFICATIONS IN A NETWORK OF NEURONS .............................................. 67  
   3.1 Approximate Homomorphisms .................................. 69  
   3.2 The Neuron Model ............................................. 70  
   3.3 The Computer Simulation ..................................... 79  
   3.4 Single Block--No Connections ................................ 81  
   3.5 Single Block--Positive Feedback ............................ 93  
   3.6 Two Blocks --A Flip-Flop .................................... 115  
   3.7 Complexity ................................................... 127  
   3.8 Conclusion ................................................... 127
IV. PROBABILISTIC HOMOMORPHISMS AND SIMPLIFICATIONS ............. 131

4.1 Probabilistic Homomorphisms ............. 131
4.2 Simplification of the Ideal Base Model ............. 135
4.3 Finite Number of Components in the Base Model ............. 142
4.4 A Probabilistic Lumped Model ............. 154
4.5 Finite Number of Neighbors ............. 165
4.6 Conclusion ............. 179

V. FUTURE DIRECTIONS ............. 180

APPENDIX ............. 184

REFERENCES ............. 188
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.1</td>
<td>X² Statistics of Base Model vs. Lumped Model (Part 1)</td>
<td>87</td>
</tr>
<tr>
<td>3.4.2</td>
<td>X² Statistics of Base Model vs. Lumped Model (Part 2)</td>
<td>88</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Second Level Chi-Square Analysis</td>
<td>89</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Base Model Unset Times from the Set State</td>
<td>124</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Minimum Input to Lumped Model to Unset It</td>
<td>125</td>
</tr>
<tr>
<td>3.6.3</td>
<td>Reset Behavior of the Base Model from the Set State</td>
<td>126</td>
</tr>
<tr>
<td>3.7.1</td>
<td>Complexity Analysis of Base Model vs. Lumped Model</td>
<td>128</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Probabilistic Lumped Stability vs. Base Model Stability</td>
<td>163</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Probabilistic Lumped Reset vs. Base Model Reset</td>
<td>164</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Synopsis of Results for Neural Network Example</td>
<td>181</td>
</tr>
</tbody>
</table>
LIST OF ILLUSTRATIONS

Figure

2.8.1 How S(t) Monotonically Decreases .................................. 43
2.8.2 An Oscillating Function .................................................. 51
3.4.1 Lumped Model Steady State Distribution \( \sigma = 10 \) .............. 84
3.4.2 Lumped Model Steady State Distribution \( \sigma = 20 \) .............. 85
3.4.3 Error Propagation .......................................................... 91
3.5.1 Lumped Model Output Behavior .......................................... 94
3.5.2 Heuristic Calculation of Lumped Steady State ....................... 101
3.5.3 Lumped Model Two Dimensional Force Field ......................... 103
3.5.4 Base Model Run 400 Neurons 50 Neighbors .......................... 104
3.5.5a Base Model Two Dimensional Run 400 Neurons 50 Neighbors ...... 106
3.5.5b Base Model Two Dimensional Run 400 Neurons 50 Neighbors ...... 107
3.5.6 Base Model Run 400 Neurons 100 Neighbors .......................... 108
3.5.7 Base Model Run 1000 Neurons 20 Neighbors ........................ 109
3.5.8 Base Model Run 1000 Neurons 30 Neighbors ........................ 110
3.5.9 Base Model Run 1000 Neurons 50 Neighbors ........................ 111
3.5.10 Base Model Run 400 Neurons 60 Neighbors ........................ 112
3.5.11 Base Model Run 800 Neurons 60 Neighbors ........................ 113
3.5.12 Base Model Run 50 Neurons 50 Neighbors .......................... 114
3.6.1 Partial Region of Stability Graph ...................................... 119
3.6.2 Entire Region of Stability Graph ....................................... 120
3.6.3 Lumped Model Two Dimensional Force Field for Flip-Flop .......... 122
3.7.1 Running Time Complexity (1) ............................................. 129
3.7.2 Running Time Complexity (2) ............................................. 130
4.4.1 Probabilistic Lumped Model Run 50 Neurons ........................ 156

vi
LIST OF APPENDICES

I. MATHEMATICAL NOTATION ........................................... 182

II. THE SIMULATION ....................................................... 184
ABSTRACT
ERGODIC MACHINES--PROBABILISTIC AND APPROXIMATE
HOMOMORPHIC SIMPLIFICATIONS

by
Sudhir Aggarwal

Chairman: Bernard P. Zeigler

Computer simulation models have been widely used to analyze complex problems. However, it is only recently that there has been an attempt to provide a formal foundation for valid modeling and simulating. One promising approach is to consider the quintuple of real world, base model, lumped model, computer, and experimental frame. The base model incorporates all structural beliefs about the "real world", whereas the lumped model is a simplified version that can be simulated on the computer.

In this dissertation, we introduce the idea of a formal deterministic model, which we have called an Ergodic Machine, to study probabilistic systems. An ergodic machine represents probabilistic features by using an ergodic transformation on a "hidden" state space. An ergodic transformation can be thought of as an ideal random number generator. The concepts of ergodic machines are developed and connections are made to Markov chains, and to random number generators.

Next, we consider a particular example of an idealized network of neurons in a formal framework. We experimentally study computer simulations of base and lumped models of this network of neurons, using
a series of computer programs developed for this purpose. We show how a deterministic lumped model can preserve information, although approximately, of a probabilistic base model.

Finally, the concepts of ergodic machines are shown to be useful in considering valid simplifications of probabilistic models. The particular example of the network of neurons is used to illustrate a valid probabilistic lumped model. This lumped model is formalized as an ergodic machine that is a homomorphic image of the ergodic machine representing the base model.

Thus, precise meaning is given to valid simplifications of probabilistic models. Hopefully, work such as this will provide the foundations necessary to allow modellers in such varied fields as business, economics, and biology to more easily construct meaningful and valid computer models.
In the approach adopted by Zeigler, the problem of validation appears quite clearly in the B → L transformation. A lumped model is valid if it generates the same input-output behavior as the base model under some experimental frame. Validation thus involves studying two formal models and how they relate.

This brings us to the question of the types of models that we wish to study. We are considering formal mathematical models as defined in Mathematical Systems Theory [Kalman, Falb and Arbib (22)] or Automata Theory [Hartmanis and Stearns (21)]. The probabilistic models that we consider are stochastic automata such as in the development by Paz [31]. In the systems theory and automata theory approach, the concept of a state variable is fundamental. Given two different models with well-defined state spaces, one can study maps between such models that preserve properties of interest. Two particular types of maps are those that preserve input-output behavior and those that preserve state transition behavior. A map from the state space of one deterministic model to the state space of the other model that preserves state transition behavior is said to be a homomorphism. Homomorphisms are natural simplifications since they preserve input-output behavior. A homomorphism is really a technique that allows us to aggregate variables and still preserve this input-output behavior within an experimental frame. This concept of a homomorphism is one that arose in the study of deterministic systems and is well-defined in such a context.

1.2 Homomorphisms

Let A and B be two deterministic systems with state sets \( S_A \), \( S_B \) and state transition functions \( \delta_A, \delta_B \). Let \( \lambda_A, \lambda_B \) be the output functions.
Then a mapping \( h: S_A \rightarrow S_B \) is said to be a homomorphism if it satisfies two conditions:

1. \( h \circ \delta_A = \delta_B \circ h \)
2. \( \lambda_A = \lambda_B \circ h \)

These two conditions allow us to show that if a homomorphism exists from \( A \rightarrow B \), then the input-output behavior of \( A \) and \( B \) are also the same. Consequently, when we consider simplifications of deterministic base models \( B \) to lumped models \( L \), we shall attempt to find a homomorphism from \( B \rightarrow L \). Unfortunately although the idea of homomorphisms has been extensively studied for the case of deterministic systems, the use of the homomorphism idea has not really been applied to probabilistic systems.

1.3 The Context

Much of the literature on probabilistic automata has concentrated on analyzing probabilistic systems in isolation, analogously to studying deterministic machines in isolation. The question that is not discussed is to what extent a deterministic system can serve as a model of a probabilistic system. Paz [31] provides a good account of the work on probabilistic automata. We find that much of the work is oriented to problems such as state minimization and problems of state equivalence. Work along such lines has been done by Carlyle [10] and Bacon [6]. Other investigations are also directly related to this, such as the work on state splitting by Santos [36].

Page [29] does consider deterministic and probabilistic sequential machines but from a different aspect. He considers the expected value of output of a probabilistic machine given a specific input string, and then defines two machines as equivalent if they have identical
expected values of output. Again, approximate homomorphic behavior is not investigated.

We analyze these problems by taking a totally different approach. Since we know much more about deterministic systems than we do about probabilistic systems, why not find a way to make use of this knowledge? We do this by introducing the concept of an ergodic machine that is a deterministic representative of a probabilistic automaton. It turns out that it is more natural to consider structural simplification of ergodic machines than it is to directly consider simplified probabilistic systems. The translation into ergodic machine terms also gives more insight into the structural behavior of probabilistic automata. The approach allows us to use existing concepts of homomorphisms and apply them directly to the ergodic machine representations.

We also discuss the simplification of a probabilistic system to a deterministic lumped model. We analyze this problem in the context of a neural network model proposed by Zeigler [44]. In this analysis, we consider an aggregation of neurons into blocks and study the relationship between this new component, the block and the original pool of neurons. Although we study the dynamics of collections of neurons, the aim is to study the lumping process. Investigations along these lines are also being done at Syracuse University. Harth, et al. [20] and Anninos, et al. [3] have explored cooperative phenomena in neural networks and have proposed an approach they call Netlet theory. Netlets are simply collections of neurons that have statistically identical properties and are connected in a uniform (though probabilistic) manner. Although the netlet model is a deterministic one, the assumptions involved in the way neurons are interconnected allow one to probabilisti-
cally predict the firing level at time $t+1$ given the firing activity of the net at time $t$. The approach adopted by this group is a detailed analysis of the netlet dynamics using the assumptions they make about the connection pattern of the neurons. The model that we use is a probabilistic one (because of assumed noise) and reduces to their model when noise is removed. We also use the lumped model explicitly to predict base model behavior.

We are in general interested in aggregating variables and forming lumped models that are more amenable to computer study than the original conceptual base model. The problem of such modelling is really a question of interest to general systems theorists. General Systems Theory [Klir (24)] however, is more interested in more unifying structural and behavioral questions rather than the validity questions in the framework of Zeigler [45]. Foo [16] has also considered validity questions in a topological investigation using the Zeigler paradigm. Corynen [11] discusses modelling and simulation in a more abstract setting than we are interested in here. The aggregation of variables to a lumped model has implicitly been done in many interesting computer simulations among which are the investigation of impulse conduction in heart muscle [Foy (17)] and the models of Mortimer [28] and Plum [32].

1.4 Thesis Organization

Chapter II introduces the concept of an ergodic machine. As mentioned before, an ergodic machine is a method to make a probabilistic system deterministic and thus make it more amenable to study. The investigation also points out the role of random number generators under simplification and what one might consider the ideal properties of such
generators. We develop the mathematical machinery that allows us to aptly represent probabilistic automata by ergodic machines.

Next, Chapter III is an investigation of the idea of using a deterministic lumped model to study a probabilistic base model. In a sense, we are studying an approximate homomorphism. The investigation is in the context of an example of a network of neurons. It is shown exactly how a great deal of information is retained by the lumped model when one investigates it properly. Limitations of the deterministic lumped model are also pointed out.

Chapter IV undertakes to apply the ergodic machine ideas to simplifications of systems. We see how naturally we are led to consider specific ergodic models that are simplified models of the base system. The simplification is presented in the context of probabilistic homomorphisms which is the concept in the probabilistic case that corresponds to the ordinary notion of homomorphism for deterministic systems.

Finally, we make some comments about future directions in Chapter V.
CHAPTER II

ERGODIC MACHINES

In this chapter, we formally give the definition of an Ergodic Machine, and develop related concepts. The terminology that we are using is that of Mathematical Systems Theory and further information on the concepts of a machine and a system may be found in Kalman et al. [22], Arbib [4] or Zadeh [40]. Intuitively, an ergodic machine is a specification for a deterministic system that has probabilistic features when viewed properly. The method by which probabilistic features are introduced is by considering part of the state space as unknown or hidden. This part of the state space will be called the seed space and one can imagine drawing elements or seeds from this seed space as analogous to choosing a pseudo random number. The method by which a new seed is chosen is by having a transformation on the seed space that defines the next element to be chosen. We shall require this transformation to be an ergodic transformation, so, before defining an ergodic machine, we first define this concept and give some examples. Billingsley [7] gives a good development of ergodic theory and the following examples are drawn from there.

2.1 Ergodic Transformations

Let \( \Omega = (\Omega; F; P) \) be a probability space (see Appendix I). A measurable transformation \( \xi: \Omega \to \Omega \) is called an ergodic transformation if it satisfies two properties: (1) \( \xi \) is measure preserving, that is, for
any $A \in \mathcal{F}$, $P(\zeta^{-1}(A)) = P(A)$, and (2) the only sets that are $\zeta$-invariant, have measure 0 or 1. A set $A$ is $\zeta$-invariant if $\zeta^{-1}(A) = A$.

An ergodic transformation thus determines a next point of the space $\Omega$, given a point $\omega \in \Omega$. Much of our development will actually depend only on $\zeta$ being measure preserving. The reasons for desiring $\zeta$ to be ergodic are partly due to the ergodic theorem described below.

**Example 2.1.1**

Let $\Omega = [0, 1)$. $\mathcal{F} = B[0,1)$, i.e. the $\sigma$-algebra of Borel Sets, also called the Borel Field, $P = dx$ where $dx$ is a Lebesque measure.

Given $\lambda \in [0,1)$ let $\zeta(x) = (x+\lambda) \mod 1$. (A number $x \mod 1$ is simply the fractional part of the number.) We shall call a transformation of this type an additive generator. It can be shown that $\zeta$ is ergodic iff $\lambda$ is irrational.

**Example 2.1.2 (the $m$-adic shift)**

$\Omega = [0,1)$, $\mathcal{F} = B[0,1)$, $P = dx$

Let $\zeta(x) = mx \mod 1$ where $m$ is a positive integer. It can be shown that this transformation is ergodic for any such $m$. Notice that if we consider the base $m$ expansion of $x$, that is $x = .w_1 w_2 w_3 \ldots$, then $\zeta(x)$ is simply $\zeta(x) = .w_2 w_3 w_4 \ldots$. This is the reason it is called the $m$-adic shift. When $m = 2$, we notice that it is a shift of the binary representation of $x$. The 2-adic shift is also called the dyadic shift. A transformation of this type will also be called a multiplicative generator.

**Example 2.1.3**

$\Omega = \{a,b,c,d,e\}$, the five point space. Let $\mathcal{F}$ be the power set of $\Omega$, that is, all subsets of $\Omega$. For each point $x \in \Omega$ let $P(x) = 1/5$. 
Let $\zeta$ be the cyclic permutation $(a,b,c,d,e)$. That is, $\zeta(a) = b$, $\zeta(b) = c$, $\ldots$, $\zeta(e) = a$. It is easily checked that $\zeta$ is measure preserving and that the only invariant sets are $\emptyset$ and $\Omega$. This example illustrates how simple an ergodic transformation may be.

2.2 The Ergodic Theorem

One of the reasons for considering ergodic transformations is because of the following powerful theorem. There are several versions of what is called an ergodic theorem and we present only a simplified version of what is better known as the pointwise ergodic theorem.

**Theorem 2.2.1**

Let $I_A$ be the indicator function of the set $A$, that is $I_A(x) = 1$ if $x \in A$ and $I_A(x) = 0$ otherwise. Then if $\zeta$ is an ergodic transformation on $(\Omega, F, P)$ and $\omega \in \Omega$

$$
\lim_{n \to \infty} \frac{1}{n} \sum_{k=0}^{n-1} I_A(\zeta^k(\omega)) = P(A) \text{ a.e. (almost everywhere).}
$$

Intuitively, the ergodic theorem simply says that if we have an ergodic transformation on a space $\Omega$ and look at a small portion of that space $A$, then the number of times that we will observe the orbit of $\omega$ in $A$ approaches $P(A)$. The orbit of $\omega$ is defined to be the set of points $\omega, \zeta(\omega), \zeta^2(\omega), \ldots$. If we consider the analogy to choosing pseudo random numbers, we would certainly wish that the long run frequency has this property. That is, if we are choosing pseudo random numbers from some uniform distribution, say $[0,1)$, then we would wish that for large $N$, the number of times a pseudo random number comes from some subinterval $[a,b)$ is approximately $N \cdot P([b-a])$ if we choose $N$ random numbers. We are now in a position to define an ergodic machine.
2.3 An Ergodic Machine

In the following, we assume that the time base is the integers.

**Definition 2.3.1**

An **Ergodic Machine** is a 7-tuple

\[ E = <X, S, \Pi, Y, \zeta, \delta_1, \lambda_1> \text{ where} \]

(a) \( X \) is the set of inputs.

(b) \( S \) is a set called the Pseudo State Set. [We shall let \( S \) be at most countably infinite.]

(c) \( \Pi = (R, F, P) \) is a probability space. \( R \) will be called the seed space and an element of \( R \) will be called a seed.

(d) \( Y \) is the output space.

(e) \( \zeta \) is an ergodic transformation on \( R \).

(f) \( \delta_1 \) is the one step transition \( \delta_1 : X \times S \times R \rightarrow S \). \( \delta_1 \) defines the next pseudo state, given the current input, the current pseudo state and the current seed.

(g) \( \lambda_1 \) is the output function \( \lambda_1 : S \times R \rightarrow Y \).

The above definition of an ergodic machine specifies a deterministic sequential machine

\[ M = <X, S \times R, \delta, Y, \lambda> \]

where \( X \) is the input space, \( Y \) is the output space and \( S \times R \) is the state space. \( \delta \) and \( \lambda \) are the transition and output functions respectively with

\[ \delta(x,s,r) = [\delta_1(x,s,r), \zeta(r)] \]

and \[ \lambda(s,r) = \lambda_1(s,r) \].

The deterministic sequential machine of course yields a system \( S_M \) when we consider an input string composed of elements of \( X \) and extend the one step transformation and output functions. Whenever we now talk
of an ergodic machine, we shall be referring to the deterministic sequential machine that the specification of an ergodic machine defines. Thus, we in particular will be talking about transformations on part of the state space of this deterministic machine.

Although we have defined the output function to be any mapping on the state space, we shall in general use only the following map.

\[ \lambda : S \times R \rightarrow Y = S \]

\[ \lambda (s,r) = s \]

That is, the output map observes as much as possible since the seed space is considered to be hidden.

Notice that although the seeds determine the next pseudo state, the next seed depends only on the present seed. The seed space essentially works autonomously, generating new values of seeds which can be used the way pseudo random numbers are.

### 2.4 Probabilistic Automata

The importance of ergodic machines derives from the fact that they can be used to deterministically model probabilistic automata. Probabilistic automata arose as a generalization of the concept of a sequential machine and Paz [31] provides an excellent reference for many aspects of this theory. Aside from giving the definition of a probabilistic automaton and considering the concept of strong lumpability, we shall make no other use of the theory of probabilistic automata.

#### Definition 2.4.1

A probabilistic automaton is a five-tuple \( < X, S, \{ Z(x) \}, Y, \lambda > \)

where

(a) \( X \) is a set of inputs.
(b) \( S \) is a set of states with \(|S|\) representing the cardinality of \( S \) or the number of elements of \( S \). We shall consider \( S \) to be at most countably infinite.

(c) \( \{Z(x)\} \) is a set of \(|S| \times |S|\) matrices, one for each input element of \( X \). The \( i,j^{th} \) entry \( Z_{ij}(x) \) represents the probability of the automaton moving to state \( j \) given that it is in state \( i \), under the input \( x \). It is clear that we must have the following conditions holding. \( \sum_j Z_{ij}(x) = 1 \) for all \( i \) for a consistent definition.

(d) \( Y \) is the output set.

(e) \( \lambda \) is the output map \( \lambda : S \to Y \).

The generalized feature of a probabilistic automaton is that instead of a definite transition from one state to the other, there is only a probability of transition. One way to study probabilistic automata is to consider starting in an initial distribution of states say \( \Pi(0) = \langle p_1, p_2, \ldots \rangle \) where \( p_i \) represents the probability of being in state \( i \) and observe with what probability one is in a particular state at time \( t \). Notice that if we consider an autonomous probabilistic automaton, that is, one with no inputs, then we exactly have the structure of a Markov Chain (see Feller [15]).

One of the concepts that naturally arises when one considers simplifications of probabilistic automata is the concept of strong lumpability. The concept of strong lumpability was first introduced in connection with finite Markov Chains by Kemeny and Snell [23].

**Definition 2.4.2**

A probabilistic automaton \( M = \langle X, S, \{Z(x)\}, Y, \lambda \rangle \) is strongly lumpable if there exists a nontrivial partition on the state set \( S \) such
that for any two states $i$ and $j$ in the same block, the sum of the row entries of $i$ and $j$ corresponding to any block are the same.

**Example 2.4.1**

Consider an autonomous probabilistic automaton $M$ with transition matrix

$$Z = \begin{bmatrix}
1/8 & 1/8 & 1/3 & 0 & 5/12 \\
1/4 & 0 & 0 & 1/3 & 5/12 \\
1/16 & 1/16 & 1/6 & 2/3 & 1/24 \\
0 & 1/8 & 1/3 & 1/2 & 1/24 \\
1/4 & 1/4 & 0 & 1/3 & 1/6
\end{bmatrix}$$

We see that the following is a partition that satisfies the definition $\langle \{1,2\}, \{3,4\}, \{5\} \rangle$.

The lumped transition matrix, that is, the one with state set the blocks of the original partition and the entries as the row entry sums, is just

$$Z' = \begin{bmatrix}
1/4 & 1/3 & 5/12 \\
1/8 & 5/6 & 1/24 \\
1/2 & 1/3 & 1/6
\end{bmatrix}$$

Intuitively, all we are doing is keeping track of the transitions from block to block. If, for any two elements of a block, the probability of transition to another block is the same, then the block to block transitions are well defined. The condition necessary to insure this is strong lumpability. We shall call the probabilistic automaton that has the blocks as states and the appropriate transition probabilities a **lumped version** of the original automaton. Thus, in the above example if $M' = \langle X', S', Z', Y', \lambda' \rangle$ where $X$ has one element and $S'$ has three elements, then $M'$ is a strongly lumped version of $M$. For the concept of strong lumpability, the output set and output transition function play no part.
2.5 Ergodic Machines and Probabilistic Automata

We have mentioned before that we shall use ergodic machines to model probabilistic automata. We explore some aspects of the connection between these two concepts in this section. The next definitions are based on the following informal discussion.

Suppose we are given an ergodic machine $M$. Then, if we are told that we are in a particular pseudo state $s_1$ and the input is $x$, and no other information, what can we say about the next pseudo state that would occur? Clearly, the next pseudo state would depend on the actual state that we are in, that is, on the hidden seed value that we happen to have as the second and hidden component of the state (i.e., we are actually in some $(s_1, r_1)$, but do not know this). Suppose instead, that we consider all the seed values that take us from $s_1$ to $s_1$ again at the next time step. That is, let $A = \{ r \in R | \delta_1(x, s_1, r) = s_1 \}$. Then, the most that we could say is that the chances of observing $s_1$ again at the next time step depend on the chances of being in $A$. But, it might be appropriate to think of the chances of being in $A$ as just the probability of the set $A$. This might be the case if, for example, we had initially chosen our initial seed with just the probability distribution induced by $P$ on $R$. (Recall that $R$ is a sample space of $\Pi = (R, F, \varphi)$. ) Then it would be reasonable to say that the probability of going from $s_1$ to $s_1$ is just $P(A)$. With these informal remarks as background, we shall define a probabilistic automata that corresponds to a given ergodic machine. We also note at this point that we are only considering ergodic machines with at most countable pseudo space for convenience.

Definition 2.5.1

Given an ergodic machine $M = \langle X, S, \Pi = (R, F, \varphi), Y, \zeta, \delta_1, \lambda_1 \rangle$, 

for any two pseudo states $s_i$, $s_j$, the $s_i$, $s_j$, $x$, transition seed space $A_{ij}(x)$, is defined to be $A_{ij}(x) = \{ r \in R | \delta_1(x, s_i, r) = s_j \}$.

Notice that for all $s_i$, $s_j$ and $x$, $A_{ij}(x)$ is just a subset of $R$.

Since we shall be interested in the probability of such sets we make the following definition.

**Definition 2.5.2**

An ergodic machine is **normal** if for all $s_i$, $s_j$, and $x$, $A_{ij}(x) \in F$ or, equivalently, $A_{ij}(x)$ is $P$-measurable.

From now on, we only consider normal ergodic machines and shall assume that ergodic machines means a normal ergodic machine.

**Definition 2.5.3**

Given an ergodic machine $M = \langle X, S, \Pi, Y, \zeta, \delta_1, \lambda_1 \rangle$, the associated automaton $C_M$ is the probabilistic automaton $C_M = \langle X, S, Y, \{ Z(x) \}, \lambda \rangle$ where $Z(x)$ is the matrix whose $i,j$th entry is:

$$Z_{i,j}(x) = P(A_{i,j}(x))$$

$\lambda$ is just the same as for the sequential machine that $M$ represents.

Observe that we have made no mention of the ergodic transformation $\zeta$. Thus, it is apparent that the same probabilistic automaton $C$ is associated with many ergodic machines. In fact, if $M$ has the associated machine $C_M$, then any ergodic machine with simply a different $\zeta$ has the same associated machine $C_M$. From the definition then, the mapping from the class of ergodic machines to the class of probabilistic automata is many to one. Furthermore, it is an easy exercise to check that this mapping is in fact surjective (onto) since given any probabilistic automata $C$ it is quite easy to find an ergodic machine $M$ such that $C = C_M$. 
Example 2.5.1

Consider an autonomous ergodic machine $M$ with

(a) $X = \{\emptyset\}$ That is, $X$ is a singleton set.

(b) $S = \{1,2,3\}$ There are three pseudo states.

(c) $R = \{a,b,c,d,e\}$ with $F$ the set of all subsets of $R$.

Let $P(\{a\}) = P(\{b\}) = \ldots = P(\{e\}) = 1/5$.

(d) Let $\zeta$ be the cyclic permutation on $R$ that is $(a,b,c,d,e)$.

(See example 2.1.3.)

(e) Let $Y = S$ and $\lambda$ be the pseudo state output.

(f) Since $M$ is autonomous, we can ignore the input in the definitions of the transition.

$\delta_1$ is defined as follows

\[ \delta_1(1, r) = 1 \quad \text{for} \quad r = a, b \]
\[ = 2 \quad \text{for} \quad r = c, d, e \]
\[ \delta_1(2, r) = 1 \quad \text{for} \quad r = a, b \]
\[ = 2 \quad \text{for} \quad r = c, d \]
\[ = 3 \quad \text{for} \quad r = e \]
\[ \delta_1(3, r) = 1 \quad \text{for} \quad r = e \]
\[ = 2 \quad \text{for} \quad r = a, d \]
\[ = 3 \quad \text{for} \quad r = b, c \]

Since $M$ is autonomous, it is apparent that the associated automaton is also autonomous and therefore simply a Markov Chain. The associated automaton has transition matrix

\[
Z = \begin{bmatrix}
2/5 & 3/5 & 0 \\
2/5 & 2/5 & 1/5 \\
1/5 & 2/5 & 2/5
\end{bmatrix}
\]

Notice that for the ergodic machine, if we start in any of the three states, there are only five different trajectories possible. (A
trajectory will be the sequence of outputs of the sequential machine
that the ergodic machine represents.) For example, starting in state 1,
we can calculate the first few elements of the five possible sequences.
If the initial state was actually (1,a) then \( \delta[(1,a)] \) is [1,b]. \( \delta[(1,b)] =
(1,c). \delta[(1,c)] = (2,d). \) Thus the sequence of outputs (in this case just
the pseudo states) is 1, 1, 1, 2. Doing this for every possible initial
state, we get the five possibilities, indexed by the initial seed.

(a) 1, 1, 1, 2, 2, 3, 2, ...
(b) 1, 1, 2, 2, 3, 2, 1, ...
(c) 1, 2, 2, 3, 2, 1, 2, ...
(d) 1, 2, 3, 2, 1, 2, 2, ...
(e) 1, 2, 1, 1, 2, 2, 3, ...

The sequences beginning with the other initial states can be simi-
larly calculated. If we did not know the initial seed but knew only that
it was one of the five seeds with equal probability, then the associated
Markov Chain would represent the probability of going from state i to j
at the first time step.

Since we are interested in using homomorphisms as simplification
procedures, it is important to understand how homomorphisms of ergodic
machines relate to the associated probabilistic automata.

**Definition 2.5.4**

Assume \( M, M' \) have the same input, output sets. An ergodic machine
\( M' \) is a homomorphic image of the ergodic machine \( M \) if there exists a homo-
morphism \( h \) (surjective) from the sequential machine that \( M \) represents to
the sequential machine that \( M' \) represents. That is, \( h:SxR \rightarrow S'xR' \) such
that (1) \( h \cdot \delta(x,s,r) = \delta'(x, h(s,r)) \) and (2) \( \lambda'[h(s,r)] = \lambda(s,r) \).

The definition of homomorphism is just the standard automata
Theorem 2.5.1

Let $h$ be a homomorphism from the ergodic machine $M = \langle X, S, \Pi, Y, \zeta, \delta, \lambda \rangle$ to the ergodic machine $M' = \langle X', S', \Pi', Y, \zeta', \delta', \lambda' \rangle$. If $h = k_1 \times k_2$ where $k_1 : S \rightarrow S'$ and $k_2 : R \rightarrow R'$ with $h(s, r) = (k_1(s), k_2(r))$, then $C_M'$ is a strongly lumped version of $C_M$.

The proof of the theorem depends on the following lemma. We shall define $\delta_1(x, b, r) = \{s \in S \mid \text{there exists } b \in B \text{ with } \delta_1(x, b, r) = s\}$. Assume the same hypotheses as for the theorem.

Lemma 2.5.1

For all $i, j \in S'$ and input $x$,

$$\delta_1(x, k_1^{-1}(i), r) \subseteq k_1^{-1}(j) \iff r \in k_2^{-1}[A_{i,j}^1(x)]$$

where $A_{i,j}^1(x)$ is just the $i,j,x$ transition seed space of the ergodic machine $M'$.

Proof:

$$(\Leftarrow)$$ Let $r \in k_2^{-1}[A_{i,j}^1(x)]$

then since $h(k_1^{-1}(i), r) = (i, k_2(r))$ with $k_2(r) \in A_{i,j}^1(x)$ we have

$$\delta'(x, k_1^{-1}(i), r) = \delta'(x, i, k_2(r)) = [j, \zeta' k_2(r)] = h\delta(x, k_1^{-1}(i), r).$$

Thus,

$$k_1 \circ \delta_1(x, k_1^{-1}(i), r) = j$$

$$\delta_1(x, k_1^{-1}(i), r) \subseteq k_1^{-1}(j).$$

$$(\Rightarrow)$$ The forward implication follows just as readily. Simply use $r \notin k_2^{-1}[A_{i,j}^1(x)]$ and show that $\delta_1(x, k_1^{-1}(i), r) \not\subseteq k_1^{-1}(j)$.

Although the above lemma is notationally complicated, it is really rather trivial and simply says that if one considers a seed $r$ that takes you from pseudo state $i$ to pseudo state $j$ in the image
machine $M'$, then, the set of seeds which are inverse images of the seed $r$
take you from the inverse image set of $i$ to the inverse image set of $j$.
Since $h$ is a homomorphism this clearly has to hold. We now present a
lemma due to Corynen [11] that we shall also use in the proof of the
theorem.

Lemma 2.5.2

Let $\zeta$ and $\zeta'$ be ergodic transformations on the spaces $(\Omega, F, P)$
and $(\Omega', F', P')$ respectively. Suppose $k$ is a surjective mapping
$k: \Omega \to \Omega'$ with $k \circ \zeta = \zeta' \circ k$. Then, $P'(A') = P'(k^{-1}(A'))$ for all $A' \in F'$.

Proof:
The proof is straightforward using the ergodic theorem we have presented
and the fact that $k \circ \zeta^j = (\zeta')^j \circ k$.

Proof (Theorem 2.5.1)

To follow the proof of the theorem the following commutative diagram
should be kept in mind (the diagram commutes because $h = k_1 \times k_2$ is a
homomorphism).

\[
\begin{array}{ccc}
X \times S \times R & \xrightarrow{\delta_1 \times \zeta} & S \times R \\
\downarrow \quad \quad \quad \downarrow \quad \quad \quad \downarrow \\
X \times S' \times R' & \xrightarrow{\delta_1' \times \zeta'} & S' \times R'
\end{array}
\]

Let $u \in k_1^{-1}(i)$ and $v \in k_1^{-1}(j)$ for $i,j \in S'$. Then, if we let
$B = k_2^{-1}(A'_1, j(x))$ it is easy to see that $A_{u,v}(x) \subseteq B$. This follows simply
from the commutativity of the above diagram, or from an application of
lemma 2.5.1. Lemma 2.5.1 further asserts that $\delta_1(x,u,B) \subseteq k_1^{-1}(j)$. Thus,
we also have

\[
\bigcup_{v \in k_1^{-1}(j)} A_{u,v}(x) = B
\]

We notice now that since $\delta_1$ is a function, the above union is, in fact,
a disjoint union. That is, \( A_{u,v}(x) \cap A_{u,v'}(x) = \emptyset \) for \( v \neq v' \) and \( v, v' \in k_1^{-1}(i) \). Now, for \( u, u' \in k_1^{-1}(i) \) we clearly have

\[
\bigcup_{v \in k_1^{-1}(j)} A_{u,v}(x) = \bigcup_{v \in k_1^{-1}(j)} A_{u',v}(x) = B
\]

\[
\therefore \quad P\left[ \bigcup_{v \in k_1^{-1}(j)} A_{u,v}(x) \right] = P\left[ \bigcup_{v \in k_1^{-1}(j)} A_{u',v}(x) \right]
\]

\[
\therefore \quad \sum_{v \in k_1^{-1}(j)} P[A_{u,v}(x)] = \sum_{v \in k_1^{-1}(j)} P[A_{u',v}(x)]
\]

(1)

for any \( u, u' \in k_1^{-1}(i) \).

Furthermore, since \( h \) is a homomorphism, \( ko\zeta = \zeta'ok \). Since \( \zeta, \zeta' \) are both ergodic transformations we can apply lemma 2.5.2 and get

\[
P'[A_{1,j}(x)] = P[B] = \sum_{v \in k_1^{-1}(j)} P[A_{u,v}(x)] \text{ for } u \in k_1^{-1}(i).
\]

(2)

Now if we let the partition of the associated automaton \( C_M \) be just the disjoint sets \( k_1^{-1}(i) \) over \( i \in S' \) it is easy to see that conditions (1) and (2) are just the strong lumpability condition that was previously defined. Thus, as asserted, \( C_M' \) is a strongly lumped version of \( C_M \).

So far, we have considered the concept of the associated automaton of an ergodic machine, without worrying about the transformation \( \zeta \) and probability space \( \Pi \). However, from example 2.5.1, it is apparent that with a simple \( \zeta \) and small sample space, we may in fact be poorly representing the associated automaton. It is clear from examining any of the fifteen possible output sequences of that example that each sequence will soon start to repeat. (Maybe it might have an initially distinct subsequence). This is so because of the fact that the sequential machine is a finite state sequential machine and it is easy to see that within a length twice the size of the state space, a cyclic subsequence can be found. We, however, wish to be able to consider more general seed spaces,
although eventually we shall of course discuss the situation for the only practical case of finite seed spaces. In the next section, we consider the concept of information related to an ergodic machine which will lead us to a better representative for the associated probabilistic automaton.

2.6 Information

We are interested in finding a \( H \) and a \( \zeta \) that will allow us to aptly represent a probabilistic automaton \( C \). Recall that the mapping from the class of ergodic machine \( M \) to the class of probabilistic automaton \( C \) was a surjective and not injective mapping. If we call the mapping \( \phi \), then we wish to explore the subclass \( \phi^{-1}(C) \) given the probabilistic automaton \( C \). Hopefully, some elements of this subclass will be such that any sequence from a proper representative ergodic machine, say \( E \), would never lead one to doubt that the ergodic machine is in fact a probabilistic automaton, if it was not known in advance that \( E \) was in fact an ergodic machine.

To simplify concepts, we consider the case of autonomous ergodic machines for the next few sections. The generalization to non-autonomous ergodic machines is quite simple and we indicate how this is done later on. Thus, from now on, ergodic machines will mean an autonomous ergodic machine, that is, one that has only a single input value. Because we consider only autonomous ergodic machines, the associated automaton will of course be simply a Markov Chain. Thus, we are searching for a proper representative of a given Markov Chain.

Suppose we are given an ergodic machine and we observe the outputs for \( n+1 \) time steps. That is, we observe the pseudo states \( s(0) \), \( s(1) \), \( s(2) \), \ldots, \( s(n) \) where each \( s(i) \in S \). We shall assume that the initial seed (unknown by us) was chosen with the distribution induced by the
measure \( P \). Thus, if \( B \subseteq R \) then the probability that the initial seed \( r(0) \in B \) is just \( P(B) \). For example, if \( R = [0,1] \) with Lebesgue measure, then the probability of the initial seed being in any interval is simply the length of the interval. We know that the ergodic machine should have trajectories that match the associated automaton. But, the fundamental feature of the associated automaton which is a Markov Chain is simply that \( P(X(n+1) = s(n+1) | s(0), \ldots, s(n)) = P(X(n+1) = s(n+1) | s(n)) \) where \( X(0), X(1), \ldots \) is a sequence of random variables whose distribution defines the Markov process.

Can we find something analogous to this for our ergodic machine? We shall see that we can, but first we need the following.

\textbf{Definition 2.6.1}

Let \( s(0), \ldots, s(n) \) be a sequence of observed pseudo states. The \textit{information gain} (IG) of this sequence is the set of seeds \( r \in R \) at time step 0 that could have given rise to the observed trajectory. That is, recalling that \( A_{i,j} \) is the \( i,j \) transition seed space, we have

\[
IG(s(0), \ldots, s(n)) = \bigcap_{i=0}^{n-1} s_i^{-1}(A_{s(i), s(i+1)})
\]

We shall abbreviate the above to \( IG(s) \) when it is clear that \( s \) means the sequence \( s(0), \ldots, s(n) \). The reason for calling \( IG(s) \) the information gain is that we initially know only that the initial seed is an element of \( R \) but after observing the output trajectory we now know that the initial seed was an element of \( IG(s) \subseteq R \).

Notice that the IG is a monotonously decreasing set as we observe more and more time steps. That is, if \( r \) is a subsequence of \( s \) then \( IG(s) \subseteq IG(r) \).

Having observed the sequence \( s(0), \ldots, s(n) \), we are interested
in what the next pseudo state could possibly be. The next pseudo state of course depends on the actual seed at time step \( n \), that is, on \( r(n) \). Thus, although we know what are the possible initial seeds we could have started from at time step 0, we need to know the possible seeds that the ergodic machine might be in at time step \( n \). This motivates the concept of the information loss.

**Definition 2.6.2**

Let \( s(0), \ldots, s(n) \) be a sequence of observed pseudo states. Then, the **information loss** (IL) of this sequence is the set of possible seeds at time step \( n \), given the observed trajectory. That is,

\[
\text{IL}[s(0), \ldots, s(n)] = \zeta^n(\mathcal{I}[s(0), \ldots, s(n)])
\]

\[
= \bigcap_{i=0}^{n-1} \zeta^{-i}(A_s(i),s(i+1))
\]

We have mentioned before that we shall assume that the chances of an initial seed \( r(0) \) being in a set \( B \) is just \( P(B) \) at time step 0. Because \( \zeta \) is a measure preserving transformation, we have the following.

**Proposition 2.6.1**

Given a measurable set \( K \subset \mathbb{R} \) at time step \( n \), the probability of \( r(n) \), the seed at time step \( n \), being in \( K \) is just \( P(K) \).

**Proof:**

\[
\text{Prob}(r(n) \in K) = \text{Prob}(r(0) \in \zeta^{-n}(K))
\]

\[
= P(\zeta^{-n}(K)) \text{ by our definition of the chances of the initial seed being in a particular set}
\]

\[
= P(K) \text{ since } \zeta \text{ is measure preserving.}
\]

We are now in a position to derive conditions so that the ergodic
machine appropriately model a Markov Chain. Given the observed sequence, $s(0), \ldots, s(n)$, we wish to investigate what we would mean by

$$\text{Prob}(s(n+1) \mid s(0), \ldots, s(n)).$$

Clearly we simply mean the chances of the seed at time step $n$ being a seed that gives rise to $s(n+1)$ at the next time step, knowing that the seed must be consistent with the observed trajectory. Thus, we first calculate the set of seeds at time step $n$ that has this property. Clearly it is just $IL(s(0), \ldots s(n)) \cap A_{s(n),s(n+1)}$. Similarly, having only observed the output at time step $n$ to be $s(n)$, the set of seeds that gives rise to $s(n+1)$ at the next time step is of course $A_{s(n),s(n+1)}$. Thus, we are justified in claiming that if $s$ represents the sequence $s(0), \ldots, s(n)$,

1. $\text{Prob}(s(n+1) \mid s(0), \ldots, s(n)) = P(IL(s) \cap A_{s(n),s(n+1)})/P(IL(s))$

2. $\text{Prob}(s(n+1) \mid s(n)) = P(A_{s(n),s(n+1)})$

The above two statements are the informal justification for the following definitions.

**Definition 2.6.3**

Given the sequence $s = s(0), \ldots, s(n)$, the information loss is **strong** at time $n$ if for all $j \in S$,

$$P(IL(s) \cap A_{s(n),j}) = P(IL(s)) \cdot P(A_{s(n),j})$$

The information loss is **weak** at time step $n$ if there is some $j$ for which the above does not hold.

**Definition 2.6.4**

Given an (autonomous) ergodic machine, $\zeta$ is a **faithful** transformation if the information loss is strong for every possible sequence at every time step.
Definition 2.6.5

Let $E$ be an (autonomous) ergodic machine for which $C_E$ is the associated probabilistic automaton, in this case, a Markov Chain. Then $E$ faithfully represents $C_E$ if $\zeta$ is a faithful transformation.

The discussion preceding the last three definitions gives ample justification for our definition of an ergodic machine faithfully representing a probabilistic automaton. Intuitively, we see that the expected trajectories of the probabilistic automaton are just the same as the expected trajectories of the ergodic machine if we assume an initial unknown seed. We further see that no more useful information about the next transition can be gotten from observing a particular sequence than can be obtained from just observing the last output. It is clear that if one were to run a faithful representative ergodic machine, an observer would never suspect that the ergodic machine is deterministic.

Proposition 2.6.2

Let $E$ be an (autonomous) ergodic machine with $\zeta$ a transformation such that for every sequence $s$, $IL(s) = R$. Then, $\zeta$ is a faithful transformation.

Proof:

It is immediate that the information loss is always strong.

Proposition 2.6.3

If the information loss of a sequence $s$ is null, that is, $IL(s) = \phi$, then the sequence $s$ could never have arisen.

Proof:

$IL(s) = \phi \Rightarrow IG(s) = \phi \Rightarrow$ no possible initial seed could give rise to the sequence $s$. 

Before continuing on, it will be helpful to get a feel for the concept of information loss and information gain by considering two examples:

Example 2.6.1

Suppose we wish to model a four-state Markov process with state set $S = \{1, 2, 3, 4\}$ and $\text{Prob}(i \rightarrow j) = 1/4$ for $i, j = 1, \ldots, 4$. Let us try to model this by a generator that is a dyadic shift on $[0, 1)$. That is, $R = [0, 1)$, $F = B[0, 1)$, $P = dx$, $\zeta(x) = 2x \mod 1$. (See example 2.1.2.) Furthermore, let the transition function be such that $A_{11} = [0, 1/4)$, $A_{12} = [1/4, 1/2)$, $A_{13} = [1/2, 3/4)$, $A_{14} = [3/4, 1)$ for $i = 1, \ldots, 4$.

Assume that we could have observed the following output sequence through time step 3.

$s(0) = 1 \quad s(1) = 3 \quad s(2) = 2 \quad s(3) = 1$

If we calculate the information gain at each time step for this sequence we get:

$IG_0 = [0, 1)$

$IG_1 = A_s(0), s(1) = A_{1,3} = [1/2, 3/4)$

$IG_2 = \zeta^{-1}[A_s(1), s(2)] \cap A_{1,3} = ([1/8, 1/4) \cup [5/8, 3/4)]) \cap [1/2, 3/4) = [5/8, 3/4])$

$IG_3 = \zeta^{-1}[\zeta^{-1}[A_s(2), s(3)] \cap A_{3,2}] \cap A_{1,3} = \zeta^{-1}[\zeta^{-1}[0, 1/4) \cap [1/4, 1/2)) [1/2, 3/4) = \zeta^{-1}(\emptyset) \cap [1/2, 3/4) = \emptyset$

$IG_i = \emptyset$ for $i > 3$ since $IG$ is monotone decreasing.

Because the information gain at time step 3 is null, we know that the observed sequence is impossible and that $\zeta$ cannot be faithful. This will become even clearer when we examine the information loss.

$IL_0 = [0, 1)$

$IL_1 = \zeta[1/2, 3/4) = [0, 1/2)$. 
\[ IL_2 = \zeta^2 I_G_2 = \zeta^2 [5/8, 3/4] = [1/2, 1] \]

\[ IL_3 = \phi \]

Notice that even at time step 1, the sequence is weak. For, we have

\[ P(IL_1 \cap A_{1,4}) = 0 \neq 1/8 = P(IL_1) \cdot P(A_{1,4}). \]

What this says is that having observed a 1 and a 3 as the output, since the possible seeds at time step 1 are only [0, 1/2), there is no way that we could possibly observe a 3 or a 4 at the next time step. Clearly, this is more than sufficient for a sequence to be weak. Of course, since \( IL_3 = \phi \), it is immediately obvious that \( \zeta \) cannot be faithful.

**Example 2.6.2**

Let everything be the same as in the previous example except that we use \( \zeta(x) = 4x \mod 1 \) as the generator. We examine the same output

\[ s(0) = 1 \quad s(1) = 3 \quad s(2) = 2 \quad s(3) = 1 \]

Let us again calculate the information gain and the information loss.

\[ IG_0 = [0, 1) \]

\[ IG_1 = A_{1,3} = [1/2, 3/4) \]

\[ IG_2 = \zeta^{-1}[1/4, 1/2) \cap [1/2, 3/4) = [9/16, 10/16) \]

\[ IG_3 = \zeta^{-2}[0, 1/4) \cap [9/16, 10/16) = [36/64, 37/64) \]

Similarly, through three time steps we can calculate

\[ IL_0 = [0, 1) \]

\[ IL_1 = \zeta[1/2, 3/4) = [0, 1) \]

\[ IL_2 = \zeta^2[9/16, 10/16) = [0, 1) \]

\[ IL_3 = \zeta^3[36/64, 37/64) = [0, 1) \]

We notice that for this sequence, the information loss is \( R = [0, 1) \) at each time step and therefore strong. If it could be shown that
for any possible sequence, the information loss was similarly $[0,1)$, then by Proposition 2.6.2 we would know that $\zeta$ was faithful. We come back to this question of showing the existence of faithful transformations in a later section.

For completeness, we present the following proposition.

**Proposition 2.6.4**

Suppose $U$ is an infinite sequence with $U_n$ being the subsequence through time step $n$. Assume $m \leq n$. Then,

1. $IG(U_n) \subset IG(U_m)$
2. $IL(U_n) \subset \zeta^{n-m}IL(U_m)$
3. $\zeta^{m-n}IL(U_n) \subset IL(U_m)$
4. $IL(U_n) = \bigcap_{i=0}^{n-1} \zeta^{-i}[A_{u(i)},u(i+1)] \subset \bigcap_{i=0}^{n-1} \zeta^{-i}[A_{u(i)},u(i+1)]$

We have previously defined the information loss function as

$$IL(s) = \bigcap_{i=0}^{k-1} \zeta^{-i}[A_{s(i)},s(i+1)]$$

for $s$ the sequence $s(0), \ldots, s(k)$. Although this definition arose naturally, it is not very convenient to work with. Fortunately, there exists an alternative definition that is often more convenient to work with. Let us call the original definition the backward information loss (BIL) and use an argument to indicate the time step through which we are considering it for a hypothesized sequence $s$. Thus,

$$BIL(k) = \bigcap_{i=0}^{k-1} \zeta^{-i}[A_{s(i)},s(i+1)]$$

Now, we can recursively define a new concept which we shall call the forward information loss (FIL).

**Definition 2.6.6**

Given a sequence $s(0), s(1), \ldots, s(n)$ the forward information
loss (FIL) is defined recursively as

1. \( FIL(0) = R \)
2. \( FIL(k) = \zeta[FIL(k-1) \cap A_s(k-1),s(k)] \) for \( 0 < k \leq n \)

Notice that \( FIL(1) = \zeta[R \cap A_s(0),s(1)] = \zeta[A_s(0),s(1)] \).

Furthermore we can represent the forward information loss explicitly as

\[ FIL(k) = \underbrace{\zeta[\ldots[\zeta[A_s(0),s(1)] \cap A_s(1),s(2)] \ldots A_s(k-1),s(k)]}_{k \zeta's} \]

we need to show that the two concepts of backward information loss and forward information loss are equivalent and to do this we need the following lemma. Notice that \( \zeta \) is not 1-1 in general.

**Lemma 2.6.1**

For any sets \( A, B \) and any transformation \( \zeta \),

\[ \zeta[\zeta[A] \cap B] \subset \zeta^2[A \cap \zeta^{-1}(B)] \]

**Proof:**

Let \( y \in \zeta[\zeta[A] \cap B] \)

Then, \( \exists x \in \zeta[A] \cap B \) such that \( \zeta(x) = y \Rightarrow x \in \zeta[A] \) and \( x \in B \).

Now, \( \exists z \in A \) such that \( \zeta(z) = x \).

Since \( x \in B \Rightarrow z \in \zeta^{-1}(B) \) and thus,

\[ z \in A \text{ and } z \in \zeta^{-1}(B) \]

\[ \Rightarrow \zeta^2(z) \in \zeta^2[A \cap \zeta^{-1}(B)] \]

\[ \Rightarrow y \in \zeta^2[A \cap \zeta^{-1}(B)] \]

The lemma can be thought of as allowing us to factor the inner \( \zeta \) through the parentheses.

**Theorem 2.6.1**

For any infinite sequence \( s(0), s(1), \ldots \)
FIL(k) = BIL(k)

Proof:

(a) We first show that FIL(k) ⊆ BIL(k)

The proof is by induction. It can trivially be shown to be true for k = 0, 1. Assume that it is true for all sequences of length k - 1.

Then,

\[ FIL(k) = \zeta[FIL(k-1) \cap A_s(k-1), s(k)] \]
\[ = \zeta[BIL(k-1) \cap A_s(k-1), s(k)] \]
\[ = \zeta \left[ \zeta^{k-1} \bigcap_{i=0}^{k-2} \zeta^{-i} [A_s(i), s(i+1)] \right] \cap A_s(k-1), s(k) \]
\[ = \zeta \left[ \zeta^{k-2} \bigcap_{i=0}^{k-2} \zeta^{-i} [A_s(i), s(i+1)] \right] \cap A_s(k-1), s(k) \]
\[ \subseteq \zeta^{2} \left[ \zeta^{k-2} \bigcap_{i=0}^{k-2} \zeta^{-i} [A_s(i), s(i+1)] \right] \cap \zeta^{-1} [A_s(k-1), s(k)] \] (by the previous lemma)
\[ \subseteq \zeta^{k} \left[ \bigcap_{i=0}^{k-2} \zeta^{-i} [A_s(i), s(i+1)] \right] \cap \zeta^{k-1} [A_s(k-1), s(k)] \] (by repeated application of the lemma)
\[ = \zeta^{k} \left[ \bigcap_{i=0}^{k-1} \zeta^{-i} [A_s(i), s(i+1)] \right] \]
\[ = BIL(k) \]

∴ FIL(k) ⊆ BIL(k).

(b) We now show that BIL(k) ⊆ FIL(k).

The proof in this direction is much easier. We simply use the fact that \( \zeta(A \cap B) \subseteq \zeta(A) \cap \zeta(B) \)

\[ BIL(k) = \zeta^{k} \bigcap_{i=0}^{k-1} \zeta^{-i} [A_s(i), s(i+1)] \]
\[ \subseteq \zeta^{k-1} \{ \zeta [A_s(0), s(1)] \bigcap_{i=1}^{k-1} \zeta^{-i+1} [A_s(i), s(i+1)] \} \]
\[
\begin{align*}
&= \kappa^{k-1} \left\{ [\kappa[A_{s}(0),s(1)] \cap A_{s}(1),s(2)] \cap \kappa^{i=2} [A_{s}(i),s(i+1)] \right\} \\
&\subset \kappa^{k-2} \left\{ [\kappa[A_{s}(0),s(1)] \cap A_{s}(1),s(2)] \cap \kappa^{i=2} [A_{s}(i),s(i+1)] \right\} \\
&= \kappa^{k-2} \left\{ [\kappa[A_{s}(0),s(1)] \cap A_{s}(1),s(2)] \cap A_{s}(2),s(3) \right\} \\
&\cap \kappa^{i=3} [A_{s}(i),s(i+1)] \\
\end{align*}
\]

We can continue in this way, distributing a \( \kappa \) through the factors to get
\[
\kappa[\kappa[ \ldots \kappa[\kappa[A_{s}(0),s(1)] \cap A_{s}(1),s(2)] \cap A_{s}(2),s(3) \ldots \cap A_{s}(k-1),s(k)] = \text{FIL}(k).
\]

Because of this theorem, we shall feel free to use the recursive definition as our definition of information loss. We now turn to a consideration of conditions that tell us when we have a faithful transformation.

2.7 Canonical Ergodic Machines and Multiplicative Generators

When we actually start defining an autonomous ergodic machine to represent a Markov Chain, it seems most natural to define the transition seed spaces as contiguous to one another if we use the space \( R = [0,1] \) as the seed space. Also, multiplicative generators seem a natural class of transformations to consider, especially when we realize that most pseudo random number generators that are implemented are either of the multiplicative type or possibly of the mixed congruential type. Knuth \([25]\) provides a good source for information on types of pseudo random number generators.

In this section we continue to consider autonomous ergodic machines, although as mentioned before, the generalization to ergodic
machines is quite simple and will be done later. Although we have allowed
the pseudo state space to be countable so far, we now wish to consider
ergodic machines for which the pseudo state space is finite. This would,
of course, correspond to finite Markov Chains as the associated proba-
bilistic automaton.

Definition 2.7.1

An ergodic machine E is semi-finite if the pseudo-state space is
a finite set.

Definition 2.7.2

An (autonomous) ergodic machine, E, is a canonical autonomous
machine if

1. \( R = [0,1) \) \( F = \text{Borel Field} \) \( P = dx \)

2. If \( A_{ij} \) is the i,j transition seed space with \( P[A_{ij}] = p_{ij} \) then if
   we define \( A_{i0} = \phi \),

   \[
   A_{ij} = \left( \sum_{k=0}^{j-1} p_{ik}, \sum_{k=0}^{j} p_{ik} \right)
   \]

Notice that we are simply requiring that the transition from
pseudo state 1 to pseudo state 1 be represented by the first interval,
from 1 to 2 be represented by the second interval, etc. In particular,
a transition from i to j is not the union of two or more noncontiguous
intervals.

The first result that we present tells us under what conditions
we have a faithful multiplicative generator for semi-finite canonical
ergodic machines. The result is useful because it establishes the fact
that faithful transformations exist.
Theorem 2.7.1

Let $E$ be a semi-finite canonical ergodic machine with pseudo state set $S$. Suppose $\zeta$ is an $m$-adic multiplicative generator, that is, $\zeta(x) = mx \pmod{1}$. Let $\mu = \min \{P[A_{ij}]\}$ for $P[A_{ij}] \neq 0$. Then if $m \geq \frac{1}{\mu}$, $\zeta$ is a faithful transformation for $E$.

Proof:

By definition of a canonical ergodic machine, given a non-null $A_{ij}$, we know that $A_{ij} = [a,b)$, that is, $A_{ij}$ is some interval.

Now, $\zeta[a,b) = \{[ma,mb)\} \pmod{1}$

We can consider mod 1 as being equivalent to wrapping the interval $[ma,mb)$ around $[0,1)$ starting at $ma \pmod{1}$.

Since $m(b-a) \geq \frac{1}{\mu}(b-a) \geq 1$ by hypothesis, we see immediately that $\zeta[a,b) = [0,1)$. Thus, if we now calculate the information loss for any sequence we have

$IL(0) = [0,1)$

$IL(1) = \zeta([0,1) \cap A_{ij}) = \zeta[A_{ij}] = [0,1)$

By induction $IL(k) = [0,1)$ for any $k$.

Notice that we have been using the forward information loss. By Proposition 2.6.2, $\zeta$ is a faithful transformation.

Theorem 2.7.2

Given any finite Markov Chain, $C$, there exists an ergodic machine $E$ such that $E$ faithfully represents $C$.

Proof:

Let $C$ be a finite Markov Chain with transition matrix $[p_{ij}]$. Let $m = \min \{p_{ij}\}$ for $p_{ij} \neq 0$. Define an ergodic machine $E$ with pseudo state set $S$ the same as the state set of the Markov Chain. Let $E$ be the canoni-
cal ergodic machine with generator \( \zeta(x) = mx \pmod{1} \). (Since we defined \( E \) to be canonical, the transition function is implicitly defined.) By the previous theorem \( \zeta \) is good for \( E \) and thus, \( E \) faithfully represents \( C \).

The importance of a canonical representation should be emphasized in Theorem 2.7.1. The following example illustrates that if we consider non-canonical machines, the theorem does not necessarily hold.

**Example 2.7.1**

Consider a very simple two-state Markov Chain as follows:

\[
[p_{ij}] = \begin{bmatrix}
1/2 & 1/2 \\
1/2 & 1/2
\end{bmatrix}
\]

Suppose we try to faithfully represent this by a dyadic multiplicative generator \( \zeta(x) = 2x \pmod{1} \) but that we use a non-canonical factorization of the seed space. That is, we let

\[
A_{11} = [0,1/4) \cap [1/2,3/4) \quad A_{12} = [1/4,1/2) \cap [3/4,1) \\
A_{21} = [0,1/2) \quad A_{22} = [1/2,1)
\]

Let us calculate the information loss function for the following potential sequence.

\[
s(0) = 1 \quad s(1) = 1 \quad s(2) = 2 \quad s(3) = 1
\]

Then, we have,

\[
IL(0) = [0,1) \\
IL(1) = \zeta([0,1) \cap A_{11}) = [0,1/2) \\
IL(2) = \zeta([0,1/2) \cap A_{12}) = [1/2,1) \\
IL(3) = \zeta([1/2,1) \cap A_{21}) = \emptyset
\]

We see that the information loss for this sequence is weak since at time step 3 we find that this sequence cannot occur. Thus, although for the canonical ergodic machine associated with the Markov Chain, the dyadic
transformation is faithful, it is not faithful for the particular non-canonical ergodic machine chosen.

The example suggests that if we are using an unknown multiplicative generator, it might be wise to model a Markov Chain by the canonical ergodic machine.

Theorem 2.7.3

Let $E$ be a semi-finite canonical ergodic machine with pseudo state $S$, $|S| = n$. Suppose that for each partition of $i$ of $[0,1)$ by the transition seed spaces, $A_{i1}, \ldots, A_{in}$ there exist at least three non-null seed spaces. Let $\zeta$ be a $m$-adic multiplicative generator, that is, $\zeta(x) = mx \pmod{1}$. Let $m = \min \{P[A_{ij}]\}$ for $P[A_{ij}] \neq 0$. Then, if $m < \frac{1}{\mu}$, $\zeta$ is not a faithful transformation.

Proof:

Suppose $P[A_{ij}] = \mu$. We can represent $A_{ij}$ by the interval $[x,y)$ since $E$ is a canonical machine. If we consider a potential sequence beginning with $s(0) = 1$, $s(1) = j$, we see that $IL(1) = \zeta([x,y))$ by definition. Now $\zeta([x,y))$ can have two forms.

1. $\zeta([x,y)) = [a,b)$ with $b-a < 1$
2. $\zeta([x,y)) = [0',c) \cup [d,1)$ with $c+d-1 < 1$

The reason that the length $b-a$ or $c+d-1$ is less than one is because of the hypothesis that $m < \frac{1}{\mu}$.

Now, since $A_{j1}, \ldots, A_{jn}$ has three non-null sets, let the first one be $A_{jf}$, a central one be $A_{jc}$ and a last one be $A_{j\ell}$.

Given case (1), $[a,b)$ must intersect $A_{jf}$, otherwise the information loss is weak, and $[a,b)$ must intersect $A_{j\ell}$, otherwise the information loss is weak. But, this implies that $[a,b)$ contains $A_{jc}$. We thus have
\[ P\{a, b \cap A_j \} = P[A_j \neq P[A_j] \cdot (b-a) \]

since \(b-a < 1\). Thus, the information loss is still weak.

The argument for case (2) is similar. In this case we find that
\([0, c)\) cannot contain all of \(A_{j,f}\) and \([d, 1)\) cannot contain all of \(A_{j,l}\). But
this implies that \(\{0, c \cup [d, 1)\} \cap A_j = \emptyset\). Again, the information loss
is weak.

The previous theorem is almost a converse of Theorem 2.7.1.

Notice that anytime we have a situation such that for all pseudo states \(i\),
there are always chances of going to three different states with positive
probability, then the condition \(m \geq \frac{1}{\mu}\) is a necessary and sufficient con-
dition for the multiplicative generator to be a faithful transformation.

We next present an example that shows why Theorem 2.7.3 is not a true
converse of Theorem 2.7.1.

**Example 2.7.2**

Suppose we consider the Markov Chain

\[
[p_{ij}] = \begin{bmatrix}
2/5 & 1/5 & 2/5 \\
1/2 & 1/2 & 0 \\
1/3 & 1/3 & 1/3
\end{bmatrix}
\]

We model this as a canonical ergodic machine with

\[
A_{11} = [0, 2/5) \quad A_{12} = [2/5, 3/5) \quad A_{13} = [3/5, 1)
\]
\[
A_{21} = [0, 1/2) \quad A_{22} = [1/2, 1) \quad A_{23} = \emptyset
\]
\[
A_{31} = [0, 1/3) \quad A_{32} = [1/3, 2/3) \quad A_{33} = [2/3, 1)
\]

Although we see that for this example \(\mu = 1/5\), it turns out that
even with \(m = 4\), the multiplicative generator \(\zeta(x) = 4x \pmod{1}\) is a
faithful transformation. Let us attempt to construct a sequence that is
weak and see where this fails. It can easily be checked that until we
get a 1 in the sequence, the information loss is always \([0, 1)\) and thus
trivially strong, since \( \zeta[0,1/2] = \zeta[1/2,1] = \zeta[0,1/3] = \zeta[1/3,2/3] = \zeta[2/3,1] = [0,1] \).

Suppose then that a 1 occurs. Without loss of generality, assume it occurs at time step 0. If a 1 or 3 occurs after the 1 at time step 0 we see that \( IL(1) = [0,1] \) and is still strong. Thus, the only possibility for a weak sequence is to have a 1 followed by a 2. Let \( s(0) = 1 \) and \( s(1) = 2 \) without loss of generality. Then \( IL(1) \) can be seen to be \( \zeta[2/5,3/5] = [0,2/5) \cup [3/5,1) \).

Notice that the information loss is not all of \([0,1)\). However, it can easily be checked that \( P[IL(1) \cap A_{21}] = P[[0,2/5)) = 2/5 = P[IL(1)] \cdot P[A_{21}] \), and \( P[IL(1) \cap A_{22}] = P[[3/5,1)) = 2/5 = P[IL(1)] \cdot P[A_{22}] \). The case with \( A_{23} \) is trivially satisfied. We see that the information loss is still strong. Now consider the possible 1 or 2 at time step 2. In either case, the information loss is \([0,1)\) since we have either \( IL(2) = \zeta[0,2/5) \) or \( IL(2) = \zeta[3/5,1) \). Thus, the information loss is strong and in particular is \([0,1)\) again. Clearly, this implies that for all sequences, the information loss is always strong, and thus we see that \( \zeta \) is a faithful transformation.

In the above example, it is worth noting that the information loss is not always the entire seed space \( R = [0,1) \). Thus, although this condition is sufficient for a faithful transformation, it is not a necessary condition. We now turn to considerations of whether or not there exist "ideal" generators.

2.8 Universally Faithful Transformations

We are now interested in investigating whether or not there are transformations that are ideal in a particular sense. For example, it would be nice if we could find a transformation \( \zeta \) such that for any
ergodic machine, $\zeta$ was a faithful transformation. Unfortunately, there
is no such transformation.

**Proposition 2.8.1**

There is no "ideally" faithful $\zeta$. That is, given $\zeta$, we can
always find an ergodic machine for which $\zeta$ is not faithful.

Proof:

Choose a set $B \subset R$ with $P(B) < 1$. Let $A_{11} = B$  let $A_{21} = \zeta^{-1}(B)$.
It is clear that the information loss is weak for the sequence $s(0) = 2$
$s(1) = 1$  $s(2) = 2$.

To ask for such an ideal or universal transformation, then, is
asking too much. Perhaps what we really need is to say that given any
probabilistic automaton $C$ and given a "universal" transformation $\zeta$,
there exists some ergodic machine $E$ (not necessarily canonical) such
that $E$ faithfully represents $C$ using the transformation $\zeta$.

**Definition 2.8.1**

A transformation $\zeta$ is *universally* faithful if, given any proba-
bilistic automaton $C$, there exists an ergodic machine $E$ such that $\zeta$ is
faithful for $E$ and $E$ faithfully represents $C$.

Although the above definition is nice, in a way it is too gen-
eral. For, in general, one only builds ergodic machines for which the
transition seed space is composed of finite unions of intervals and for
which the transformation has some reasonable properties such as being
bounded or continuous.

We shall show that transformations that are multiplicative and
additive, among others, are not universal, but first we need to develop
some machinery.
Lemma 2.8.1

Consider an ergodic machine with a positive measure transition seed space $A_{11}$. Then, if we consider the information loss function corresponding to the sequence 1,1,1,1, ..., we have

$$IL(k) \supset IL(k+1) \quad \text{for } k = 0,1,2, \ldots$$

Proof:

The proof is trivial by induction.

Clearly, $IL(1) \subset IL(0) = R$

Assume true for $k-1$, we get

$$IL(k+1) = \zeta[IL(k) \cap A_{11}] \subset \zeta[IL(k-1) \cap A_{11}] = IL(k)$$

Definition 2.8.2

A transformation $\zeta$ is bounded if there exists some $M$ such that $P[\zeta(A)] \leq M \cdot P[A]$ for all sets $A \subset R$. $M$ is the bound of the transformation.

Lemma 2.8.2

Let $\zeta$ be a bounded transformation with bound $M$, and consider an ergodic machine, $E$, for which $P[A_{11}] > 0$. Consider the information loss for the sequence 1,1,1,1, ... . Then (1) either $\lim_{k \to \infty} P[IL(k)] = 0$ or (2) $\frac{1}{M} \leq P[A_{11}]$.

Proof:

Since the information loss is monotonically decreasing by the previous lemma either condition (1) has to hold, or there exists a set $K$ with $P(K) > 0$ such that $\lim_{k \to \infty} IL(k) = K$. Assuming there exists such a set $K$, we see that it must be that $K = \zeta[K \cap A_{11}]$, that is, if we let $B = K \cap A_{11}$, $B$ is a subset of $A_{11}$ that expands under $\zeta$ to the entire set $K$. 
Now, for the information loss to be strong, we must have
\[
\frac{P[K \cap A_{11}]}{P(K)} = P[A_{11}]
\]
Now, since \( \zeta \) is bounded by \( M \),
\[
\frac{1}{M} \leq \frac{P[A]}{P[\zeta(A)]} \quad \text{for any set } A \text{ with } P[A] > 0.
\]
Since \( B = K \cap A_{11} \) and \( \zeta(B) = K \) we have
\[
\frac{1}{M} \leq \frac{P[B]}{P[\zeta(B)]} = P[A_{11}]
\]
\[
\Rightarrow \frac{1}{M} \leq P[A_{11}]
\]
Thus, the second condition must be satisfied.

**Theorem 2.8.1**

Let \( \zeta \) be a bounded universally faithful generator.

Then \( \lim_{k \to \infty} P[IL(k)] = 0 \)

**Proof:**

By the previous lemma, either (1) \( \lim_{k \to \infty} P[IL(k)] = 0 \) or
(2) \( \frac{1}{M} \leq P[A_{11}] \). The second case cannot be true for a universally faithful generator, since then we could not model a probabilistic automaton which had the probability of transition from state 1 to 1, \( p_{11} < \frac{1}{M} \).
Thus, the first case must hold.

Notice that this theorem does not assert the existence of universally faithful bounded generators. It only says that if one does exist, it must require \( \lim_{k \to \infty} P[IL(k)] = 0 \).

We now consider a sequence of lemmas that investigates what happens when the probability of the information loss of a sequence 1,1,1, ... approaches 0. We have shown already that the second condition in the previous lemma will exclude universal transformations. From now on, we
shall consider the case where the seed space is \( R = [0, 1) \) and where any transition seed space is composed of a finite union of intervals. We first present some notation that we shall find convenient to use.

**Definition 2.8.3**

Given the sequence 1,1,1, ... we let \( S(t) = IL(t) \cap A_{11} \). That is, \( S(t) \) is the part of \( IL(t) \) that is a subset of \( A_{11} \). Notice that \( S(t) \) is monotonically decreasing and that \( IL(t+1) = \zeta[S(t)] \). Also, \( S(0) = A_1 \).

**Definition 2.8.4**

Let \( Y \) be some interval in \( R = [0, 1) \). \( Y \) will be called the **target region**. Let \( X \) be a union of intervals \( \{X_i\} \) that contains \( \zeta^{-1}(Y) \). \( X \) will be called the **initial region**. Furthermore, define

\[
IR(t) = X \cap S(t) \quad t \geq 0
\]

\[
TR(t) = Y \cap S(t) \quad t \geq 0
\]

\( IR(t) \) is the initial region at time step \( t \) and is part of the initial region that is also part of \( S(t) \). The interpretation for \( TR(t) \) is similar. In the application of the following two lemmas, it turns out that we only use the entire interval \([0, 1)\) as the target region. However, since this special case does not particularly simplify the proofs, we consider an arbitrary target region in the two lemmas.

Figure 2.8.1 below is a pictorial representation of how \( S(t) \) decreases with time. Notice that at least initially, the number of intervals of \( S(t) \) in the target region may increase.

**Definition 2.8.5**

Let a set \( A = \bigcup_{i=1}^{n} I_i \) where \( I_i \) is an interval. Then \( \# : \{\text{Finite unions of intervals}\} \to \text{Integers} \) by \( \#(A) = n \). \# counts the number of
finite intervals that compose a set $A$. It can easily be checked that
\[ \sum_{i=1}^{n} \# A_i \leq \sum_{i=1}^{n} \#(A_i). \]

**Lemma 2.8.3**

$$\text{TR}(t+1) = \zeta[\text{IR}(t)] \cap \text{TR}(t)$$

**Proof:**

$$\text{TR}(t+1) = \text{TR}(t) \cap Y \cap S(t+1) \quad \{\text{TR}(t) \text{ monotonically decreasing}\}$$

$$= \text{TR}(t) \cap Y \cap \zeta[S(t)] \cap A_{11} \quad \{\text{using definition of } S(t)\}$$

$$= \text{TR}(t) \cap \zeta[S(t)] \quad \{\text{since } TR(t) \subseteq TR(0) = Y \cap A_{11}\}$$

Now, by the definition of the initial region $X$, we have

$$S(t) = \text{IR}(t) \cup [\bar{X} \cap S(t)] \quad \{\text{bar represents set complement}\}$$

but, \(\zeta[\bar{X}] \cap Y = \emptyset \Rightarrow \zeta[\bar{X}] \cap \text{TR}(t) = \emptyset\)

\[ . \cdot . \quad [S(t)] \cap \text{TR}(t) = \{\zeta[\text{IR}(t)] \cap \text{TR}(t)\} \cup \{\zeta[\bar{X} \cap S(t)] \cap \text{TR}(t)\} \]

\[ = \zeta[\text{IR}(t)] \cap \text{TR}(t) \cup \emptyset \]

\[ . \cdot . \quad \text{TR}(t+1) = \zeta[\text{IR}(t)] \cap \text{TR}(t) \]
We now come to the crucial lemma. This lemma will be used to show the limitations of transformations for which IL(k) → ϕ.

An intuitive interpretation of the following lemma is in order. Suppose we consider the transition seed spaces $A_{11}, \ldots, A_{1n}$. Then, if we consider the information loss for a sequence 1,1,1,1, \ldots we know by hypothesis that the probability of the information loss decreases to 0. However, for the information loss to be strong we see that $\zeta[IL(t) \cap A_{11}]$ must always intersect each $A_{11}, \ldots, A_{1n}$ non-trivially. We have defined $IL(t) \cap A_{11}$ to be just $S(t)$ so this says that $\zeta[S(t)]$ must always intersect each $A_{11}, \ldots, A_{1n}$ non-trivially. Clearly since the size of the intervals is shrinking we are getting smaller and smaller image intervals in any region. See Fig. 2.8.1. The following lemma in a sense puts a limit on how many intervals can be spread throughout the space $[0,1)$ as images of $S(t)$.

**Lemma 2.8.4**

Let $\zeta$ be a bounded generator, with bound $M$, on $[0,1)$ that maps intervals into intervals assuming wrapping around $[0,1)$. Suppose that $\lim_{t \to 0} P[S(t)] = 0$. Then, given any target set $Y$, there exists $t_1$ such that for all $t_2 \geq t_1$

$$\#[IR(t_2)] = j \Rightarrow \#[TR(t_2+1)] \leq j$$

**Proof:**

Suppose that $A_{11} = \bigcup_{i=1}^{n} I_i$ where $I_i = [a_i, b_i]$

Consider $\epsilon = \frac{1}{M} \min\{(a_{i+1} - b_i)_{i=1}^{n-1}, a_1 + 1 - b_n\}$

That is, we consider $\frac{1}{M}$ times the minimum distance from one interval to the next, assuming we are wrapping around $[0,1)$. We also assume that all such unions of intervals such as $A_{11} = \bigcup_{i=1}^{n} I_i$ are proper, that is, for no
i, j, is \( I_i \cup I_j \) an interval.

Let \( t_1 \) be such that \( P[S(t)] < \varepsilon \). In particular, then, an interval of \( S(t) \) can never span the region between \( I_i \) and \( I_{i+1} \).

Now, since \( IR(t) \) and \( TR(t) \) are always composed of finite unions of intervals, we let

\[
IR(t_2) = \bigcup_{i=1}^{j} J_i
\]
\[
TR(t_2) = \bigcup_{i=1}^{\ell} L_i
\]

We claim that \( \zeta[J_1] \) cannot intersect non-trivially with more than one \( L_i \). [Remember that \( \zeta[J_1] \) cannot span two intervals of the original \( A_{11} \).] For, suppose \( \zeta[J_1] \) did intersect with more than one \( L_i \). Then, since \( \zeta[J_1] \) is an interval, we must have, for some \( i' \),

\[
L_{i'} = [x_1, x_2]
\]
\[
L_{i'+1} = [x_3, x_4]
\]
\[
x_3 > x_2
\]
\[
\zeta[J_1] = [y_1, y_2] \) with \( y_1 < x_2 \) and \( y_2 > x_3 \).

But now, consider the region \([x_2, x_3]\) which is not part of \( TR(t_2) \).

Since \( \zeta[J_1] \) spans this region, and since \( IR(t) \) is monotonically decreasing, for every \( t < t_1 \), some interval of \( IR(t) \) must have spanned \([x_2, x_3]\). Furthermore, since \([x_2, x_3]\) must have been part of \( A_{11} \), [we have already eliminated spanning the region between two intervals of \( A_{11} \)], we see that \([x_2, x_3]\) is contained in \( TR(t_2) \). But, this is a contradiction, since we assumed that \( \bigcup_{i=1}^{\ell} L_i \) is a proper union. Thus, each \([J_1]\) intersects non-trivially with at most one \( L_i \), of \( TR(t_2) \).

Now, we have,

\[
# [TR(t_2+1)] = \# [\zeta[IR(t_2)] \cap TR(t_2)] \quad \text{by lemma 2.8.3,}
\]
\[
= \# [\zeta[\bigcup_{i=1}^{j} J_i] \cap \bigcup_{i=1}^{\ell} L_i]
\]
\[= \sum_{i=1}^{j} \sum_{i'=1}^{L} \sum_{i=1}^{L} \sum_{i'=1}^{L} \#\left( \mathcal{J} \cap \mathcal{L} \right) \]

This completes the proof of the lemma.

We have seen how the lemma limits the number of image intervals of \( S(t) \). The following theorem shows that we in fact have succeeded in limiting the number of intervals that can arise in the target region \([0,1)\). Recall from fig. 2.8.1 that this is limiting the number of hatched regions that can develop.

**Theorem 2.8.2**

Given the hypotheses of the previous lemma, there is an upper bound to the number of intervals of \( S(t) \). That is, for some \( L \),

\[\lim_{t \to \infty} \#\left( S(t) \right) \leq M \]

**Proof:**

Let us consider the special case in which the target region is \([0,1)\). Then, we can consider \([0,1)\) to be the initial region as well.

By the previous lemma, \( \exists t_1 \) such that for \( t_2 \geq t_1 \),

\[\#\left( \text{IR}(t_2) \right) = j \quad \#\left( \text{TR}(t_2+1) \right) \leq j \]

Let \( \#\left( \text{IR}(t_1) \right) = L \). Then since we have \( \text{IR}(t) = S(t) \) and \( \text{TR}(t) = S(t) \) for this case we have \( \text{IR}(t_1+1) = \text{TR}(t_1+1) \leq L \). That is, \( S(t_1+1) = \text{IR}(t_1+1) \leq \text{IR}(t_1) = L \). By induction, it follows that for any \( k \geq 0 \)

\[S(t_1+k) \leq L \]

Notice that any generator that is continuous on a metric space maps intervals into intervals. Furthermore, both multiplicative and additive generators are bounded. In particular, the hypotheses of the previous lemma and theorem apply to generators of the form \( \zeta(x) = mx+b \mod n \) for positive integers \( m \) and \( n \).
We now know that if $P[S(t)] \to 0$ that the number of intervals that it can be composed of is at most $L$ for some $L$. We shall use this fact to help us in showing that the information loss in such cases can never be strong for all sequences. Notice of course that $L$ depends on the initial partition of the transition seed spaces. Thus, we cannot simply say that considering an ergodic machine with a large number of states, the information loss is weak since $[S(t)]$ can never span all of the transition seed spaces. However, we do notice that $S(t)$ is eventually located at very few places. That is,

**Lemma 2.8.5**

Let $\zeta$ be a bounded generator, with bound $M$ on $[0,1)$ and let $\lim_{t \to \infty} P[S(t)] = 0$. Then $\lim_{t \to \infty} S(t) = K$ where $K$ is a set of at most $L$ distinct points, and $L$ is as in Theorem 2.8.2.

**Proof:**

Each interval of $S(t)$ decreases in length. Since eventually, by Theorem 2.8.2 there are at most $L$ intervals, each interval always contains at most one point within it, as $t \to \infty$. Otherwise, the interval would have to divide into two, creating more than $L$ intervals. Thus, there can be at most $L$ points which are in $S(t)$ for every $t$.

**Definition 2.8.6**

If $\lim_{t \to \infty} S(t) = K$, where $K$ has at most $L$ points, then a point of $K$ will be called a **final point** of $S(t)$.

We are now in a position to give any easy proof for the fact that an additive generator cannot be universally faithful for ergodic machines whose transition seed spaces are composed of finite unions of intervals and seed space is $[0,1)$. Later on, we shall prove a more general theorem.
for which an additive generator will be a special case, but the special proof for the additive case is illuminating.

**Lemma 2.8.6**

Let $\zeta$ be a faithful additive generator on $[0,1)$ and let $0 < P[A_{11}] < 1$. Then, for the sequence 1,1,1, ..., $S(t)$ has $L$ final points for some $L$.

**Proof:**

By definition, $\zeta$ is measure preserving. Also, since $\zeta$ is additive, $\zeta$ is 1-1. Thus, it is easy to see that $P[\zeta(A)] = P[A]$ for any set $A$. Now, for $\zeta$ to be faithful, we require that

$$ P[IL(k) \cap A_{11}] = P[IL(k)] \cdot P[A_{11}] $$

Thus, corresponding to the sequence 1,1,1, ..., we have

$$ P[IL(k)] = P[\zeta[IL(k-1) \cap A_{11}]] = P[IL(k-1) \cap A_{11}] = P[IL(k-1)] \cdot P[A_{11}] $$

Since $P[IL(1)] = P[A_{11}]$, we have that $\lim_{k \to \infty} P[IL(k)] = \lim_{k \to \infty} \left( P[A_{11}] \right)^k = 0$

Since $S(t) = IL(t) \cap A_{11}$, we have $\lim_{t \to \infty} P[S(t)] = 0$.

By **Lemma 2.8.5**, the conclusion follows.

**Lemma 2.8.7**

Let $\zeta$ be a faithful additive generator on $[0,1)$ and let $0 < P[A_{11}] < 1$. Then, the $L$ final points of $S(t)$ can be partitioned into disjoint sets such that within each set, the points can be ordered so that $x_{i+1} - x_i = \alpha \ [\text{mod} \ 1]$, where $\alpha$ is the additive constant of the generator. (The disjoint sets will be called cycles.)

**Proof:**

Suppose for some final point, which without loss of generality we call $x_L$, there is no point $x_i$ with $x_i + \alpha = x_L \ [\text{mod} \ 1]$. In fact, let $x_i + \beta_i = x_L$ for $i = 1, ..., L-1$, with $\beta_i \neq \alpha$. Since $P[S(t)] \to 0$, we can
choose a $t_0$ so that each interval of $S(t_0)$ has length less than $c = \min_{1 \leq i \leq L-1} \frac{1}{2} \beta_{i-1} - \alpha$. It is clear that if $I_i(t_0)$ is the interval containing $x_i$, then $\zeta I_i(t_0)$ is disjoint from $I_L(t_0)$. Consequently, $\zeta[S(t_0)]$ is disjoint from $I_L(t_0) \Rightarrow x_L$ cannot be a final point of $S(t)$.

**Proposition 2.8.2**

If there are $L$ final points corresponding to an additive generator on $[0,1)$, $L > 0$ then if $\alpha$ is the additive constant $\alpha$ is rational.

**Proof:**

Since the $L$ final points occur in cycles by the previous lemma for some integer $I$, we have $I\alpha = 1 \pmod{1} \Rightarrow \alpha$ is rational.

**Theorem 2.8.3**

An additive transformation $\zeta(x) = x + \alpha \pmod{1}$ on $R = [0,1)$ is not universally faithful for ergodic machines with $R = [0,1)$ and whose transition seed spaces are finite unions of intervals.

**Proof:**

We can choose a probabilistic automaton to model such that $0 < P[A_{11}] < 1$. Now, we know that $P[S(t)] \rightarrow 0$, and thus, there are at most $L$ final points of $S(t)$. But, since $\zeta$ is an ergodic transformation, $\alpha$ must be irrational $\Rightarrow L = 0$. Thus, after some time, $I_L(t) = \phi$. But $I_L(t) = \phi \Rightarrow$ information loss is weak for the sequence $1,1,1,\ldots$.

$\Rightarrow \zeta$ is not faithful.

The previous theorem actually says more than just the fact that an additive generator is not universally faithful given a few reasonable restrictions. It in fact shows that for any ergodic machine with $0 < P[A_{11}] < 1$, $\zeta$ is not faithful. Thus, this in effect shows that additive transformations are very poor transformations since they can only
trivially faithfully represent probabilistic automata. We can now proceed to prove a more general theorem that will show, in particular, that multiplicative transformations on [0,1) are not universally faithful.

Proposition 2.8.3

If P[S(t)] → 0 for a bounded transformation ζ, that maps intervals to intervals, then the L final points of S(t) can be partitioned into disjoint cycles, such that within each cycle there is an ordering of the points, say x₁, ..., xₖ, such that ζ(xᵢ) = xᵢ₊₁ for i = 1, ..., k-1 and ζ(xₖ) = x₁.

Proof:

It is immediate that for any final point xⱼ, there must be at least one final point xᵢ such that ζ(xᵢ) = xⱼ. The proof of this fact is exactly as the proof of Lemma 2.8.7. Call xᵢ the predecessor to xⱼ and xⱼ the successor to xᵢ. The proof will be complete when we show there is exactly 1 predecessor. But this is trivial, for we know there are L successors and thus exactly L predecessors. Since each final point must have at least 1 predecessor, it can have at most 1 predecessor.

Lemma 2.8.8

Let ζ be a bounded transformation on [0,1) that maps intervals to intervals. Let x₁, ..., xₖ be the L final points of S(t) and assume P[S(t)] → 0. Then if for all xᵢ, xᵢ is not a point of closure of Aᵢⱼ, then ζ[S(t₀)] ∩ Aᵢⱼ = ∅ for some t₀.

Proof:

Since xᵢ is not a point of closure of Aᵢⱼ for all i, 
\[ \min_i \{d(xᵢ, Aᵢⱼ)\} = \varepsilon > 0, \] where d(xᵢ, p) = |xᵢ - p| for any point p and the distance to a set is the natural generalization (remember that [0,1) is
a metric space). Now, choose $t_0$ such that the length of each interval of $S(t) < \varepsilon / M$ where $M$ is the bound of the generator and such that there are only $L$ intervals of $S(t)$. Now, $\zeta[S(t_0)]$ can only cover the points $x_1, \ldots, x_L$ and each interval covering $x_i$ has length less than $\varepsilon \Rightarrow [S(t)]$ is disjoint from $A_{ij}$. Thus, $\zeta[S(t)] \cap A_{ij} = \phi$.

Definition 2.8.7

A transformation $\zeta$ oscillates if there exists some point $x$ such that (1) for all $h$, $\zeta$ on $(h,x]$ is not monotonically increasing or decreasing or (2) for all $h$, $\zeta$ on $[x,h)$ is not monotonically increasing or decreasing.

Notice that if a transformation doesn't oscillate, this is equivalent to saying that for an $h$ sufficiently close to any point $x$, if the image of $x$ is $x'$, then $\zeta([x,h]) = [x',h')$ or $(h',x')$ for some $h'$ and similarly for $(h,x]$.

An example of a function that oscillates is $\zeta(x) = x \sin \frac{1}{x}$, $\zeta(0) = 0$. From fig. 2.8.2 below we see that the function oscillates at 0.

![Fig. 2.8.2 An Oscillating Function](image)

It is clear that for most generators that one would think of, they would be non-oscillating. It is certainly true that multiplicative and additive generators are non-oscillating.
Theorem 2.8.4

Let $\zeta$ be a bounded transformation that maps intervals into intervals and suppose that $\zeta$ is non-oscillating. Assume $0 < P[A_{11}], P[A_{12}] < 1$ and that $\lim_{t \to \infty} P[S(t)] \to 0$. Then, $\zeta$ is not a faithful transformation.

Proof:

By a previous lemma, there are $M$ final points of $S(t)$. Furthermore, these points are on disjoint cycles. Since $P[A_{12}] > 0$, we see that some final point must be a point of closure of $A_{12}$. Otherwise, the transformation loss is immediately weak for an appropriate sequence. Now, let $x$ be a final point that is a point of closure of $A_{12}$. Let the sequence of final points on the cycle of $x$ be $x = x_1, x_2, \ldots, x_n = \bar{x}$ where $\bar{x}$ is the predecessor to $x$. We assume that since $P[S(t)] \to 0$, we look at $S(t)$ for $t$ such that any interval of $S(t)$ containing an $x_i$ satisfies the non-oscillating hypothesis. That is, in particular, the image of $[x_i, h]$ is either $(h_1, x_{i+1}]$ or $[x_{i+1}, h_2)$.

The following diagram will be helpful.

\[
\begin{array}{c}
\bar{x} \\
| \\
| \\
| \\
x
\end{array}
\]

Part of $A_{11}$ \hspace{1cm} Part of $A_{12}$

Since $x$ is a point of closure of $A_{12}$ and a final point of $A_{12}$, we have two possible cases.

1. $(z, x] \in A_{11}$ and $(x, y] \in A_{12}$ for $z < x < y$ or
2. $[y, x) \in A_{12}$ and $[x, z) \in A_{11}$ for $y < x < z$.

We assume case (1) as in the diagram above. The other case is handled similarly. Let us consider the part of $S(t)$ that contains $\bar{x}$.

Suppose it is an interval $(a, b)$ that properly contains $\bar{x}$. It can be seen
that since we assume that $S(t)$ is such that $\zeta$ is monotonically increasing or decreasing on each interval containing $x_1$, after at most $n$ time steps, the part of $S(t+n)$ that contains $\bar{x}$ must be either $[\bar{x}, h_n]$ or $(h_n, \bar{x}]$ for some $h_n$. In either case, the image of this set, call it $I$, must either lie entirely in $A_{11}$ or be the single point $x$. If it is the single point $x$, then after $n$ more time steps, $S(t)$ is only composed of at most $n$ points and thus the information loss will be weak. In the contrary case, $\zeta[I] \cap A_{12} = \emptyset$. Consequently, since this can be done for every interval of $A_{12}$ (recall that there are only a finite number), for some $t_0$,

$$IL(t_0) \cap A_{12} = \emptyset$$

$\Rightarrow \zeta$ is not faithful.

**Corollary 2.8.1**

Suppose $\zeta$ is a bounded, non-oscillating transformation on $[0,1)$ that maps intervals to intervals. Then, $\zeta$ is not universally faithful for ergodic machines with $R = [0,1)$ and whose transition seed spaces are the finite unions of intervals.

*Proof:*

Theorem 2.8.1 and Theorem 2.8.4

**Corollary 2.8.2**

A transformation $\zeta(x) = mx+b \pmod{1}$ is not universally faithful for ergodic machines with $R = [0,1)$ and whose transition seed spaces are the finite union of intervals.

*Proof:*

It is each to check that $\zeta$ satisfies the hypotheses of Corollary 2.8.1.
Although further results on universally faithful transformations could be derived, the results of this section are adequate for our purposes. We now turn to a consideration of the relationship between pseudo-random number generators and representative ergodic machines.

2.9 Pseudo-Random Number Generators

We have investigated ergodic transformations on $[0,1)$ of the multiplicative and additive type. These transformations are similar to actual pseudo-random number generators commonly used. For example, a mixed congruential method is one in which $n_{i+1} = (an_i + b) \pmod{m}$ where $n_0$ is the initial seed, $a$ is the multiplier, $b$ is the additive constant and $m$ is often taken to be the word size of the computer. The sequence $\{n_i\}$ is, of course, the sequence of pseudo-random numbers generated. Since we have been working mainly on the interval $[0,1)$, it would be nice to have some way of translating results on $[0,1)$ to a more general interval $[0,m)$. The following result is useful along these lines.

Proposition 2.9.1

A transformation $\zeta(x) = [ax+b] \pmod{m}$ on the space $([0,m), B[0,m), P = \frac{1}{m} \, dx)$ is isomorphic to the transformation $\zeta'(x') = [ax' + b/m] \pmod{1}$ on the space $([0,1), B[0,1), P = dx)$.

Proof:

Consider the mapping $h$: $[x] \pmod{m} \to [x/m] \pmod{1}$. For convenience we will use $[x]_m$ to mean $[x] \pmod{m}$. Then, it is easy to check that $h$ is well defined. Furthermore, $h$ is bijective (1-1 & onto). We need only check that the following diagram commutes.
Now, $h[\zeta([x]_m)] = h([ax+b]_m) = \left[ \frac{ax+b}{m} \right]_1$

and $\zeta'[h([x]_m)] = \zeta'[\left[ \frac{x}{m} \right]_1] = \left[ \frac{x}{m} + \frac{b}{m} \right]_1 = \left[ \frac{ax+b}{m} \right]_1$

For example, a commonly used generator is the multiplicative generator $T(x) = 5^{13}x \pmod{2^{35}}$. One way of investigating what our results yield in the case of actual generators is to consider the interval $[0,2^{35})$.

If we had a generator on the entire interval we know from our previous work that for canonical ergodic machines, the generator would be faithful as long as the probability of transition from one state to the other was at least $1/5^{13}$. Clearly, this result on the interval $[0,2^{35})$ would still make sense if we considered only integral points in $[0,2^{35})$ as the only positive points that could arise. Thus, limitations of generators on the entire interval $[0,2^{35})$ would still hold if we considered only integral points in this interval. We would thus make the observation that to have a successful multiplicative generator the multiplicative constant should be fairly large. This is a theoretical result that has been seen to be true experimentally.

The main problem in studying actual pseudo random number generators is that the seed space in this case is finite. For a finite seed space, however, it is easy to see that an ergodic transformation for an ergodic machine is never faithful if the ergodic machine is nontrivial. There are two ways to see this. First, observe that if the seed space is finite and the pseudo state space is finite, we have a finite
deterministic automaton, then one can always find a repeating cycle within a length twice the number of states of the automaton (we are looking at the autonomous case). Secondly, assuming the transition from 1 to 1 is nontrivial, that is \(0 < P[A_{11}] < 1\), for a finite seed space it is straightforward to see that the information loss of the sequence 1,1,1, ... becomes \(\phi\) after some time and thus the information loss is not strong. Consequently, \(\zeta\) on a finite seed space could not be faithful. These remarks give us the following.

**Proposition 2.9.2**

If the seed space is finite, then the ergodic machine \(E\) will not faithfully represent its associated automaton unless the associated automaton is trivial. [For example, has 1 state.]

The information that we can directly apply to pseudo random number generators is thus somewhat limited. We now wish to turn to the question of considering transformations that act on \(R^N\) extrapolating from transformations operating on \(R\).

**2.10 Parallel Seeds and Permutation Independence**

So far, we have been considering pseudo space \(S\) and a probability space \(\mathcal{H} = (R, \mathcal{F}, P)\) with an ergodic transformation \(\zeta\) acting on this space. The situation can be thought of as investigating a single stream of pseudo random numbers used to effect certain probabilistic transitions. However, it often happens that we wish to consider not \(S\) but an \(N\)-fold product of \(S\) which we call \(S^N\), and rather than considering \(R\) we wish to consider \(R^N\). Naturally, since both these spaces can be used for the pseudo state space and seed space respectively, previous theory applies to an ergodic transformation on this space. But suppose the
transformation on \( R^N \) is to be just an \( N \)-fold product of the transformation \( \zeta \). That is, given a seed on \( R^N \) which will now of course be a vector, \(<r_1, \ldots, r_N>\), assume that the transformation \( \zeta \) on \( R^N \) is just
\[
\zeta<r_1 \ldots r_N> = \langle \zeta(r_1), \zeta(r_2), \ldots, \zeta(r_N) \rangle
\]
where \( \zeta \) is a transformation on \( R \). What can we now say about \( \zeta \) and what can we say about the resulting machine whose pseudo state is a vector \(<s_1, \ldots, s_N>\)?

The first consideration that we must handle is what is meant by the cross-product of two ergodic machines.

**Definition 2.10.1**

Let \( E \) and \( E' \) be two ergodic machines \( E = \langle X, S, \Pi, Y, \zeta, \delta_1, \lambda_1 \rangle \) and \( E' = \langle X', S', \Pi', Y', \zeta', \delta', \lambda' \rangle \). Then the cross-product machine is defined to be \( \langle XX', SXS', \Pi \Pi', YXY', \zeta \chi \zeta', \delta_1 \chi \delta_1', \lambda_1 \chi \lambda_1' \rangle \) where \( \Pi \Pi' \) is the product space \( (R \times R', \bar{F}, \bar{P}) \) with \( \bar{P} \) the product measure such that for measurable sets \( A \) and \( A' \) in \( R, R' \) respectively \( \bar{P}[AxA'] = P[A] \cdot P'[A'] \) and \( \bar{F} \) is the smallest \( \sigma \)-algebra that contains all sets of the form \( AxA' \) where \( A \) and \( A' \) are in \( F \) and \( F' \) respectively. The transformation \( \zeta \chi \zeta' \), \( \delta_1 \chi \delta_1' \), and \( \lambda_1 \chi \lambda_1' \) of course, act componentwise. An \( N \)-fold cross-product machine is the obvious generalization.

Notice that although an \( N \)-fold cross-product machine is specified just like an ergodic machine, the necessary condition for it to be an ergodic machine is that \( \bigwedge_{i=1}^{N} \zeta_i \) be an ergodic transformation on \( R^N \).

This is not true in general if \( \{ \zeta_i \} \) are ergodic transformations, for we can consider the following example.

**Example 2.10.1**

Let \( \zeta \) be the ergodic transformation on \( R = \{a, b, c, d, e\} \) that is cyclic. [See example 2.1.3.] Consider \( \zeta \chi \zeta \) on \( R^2 \). It is obvious that
\( \zeta \chi \zeta \) is not ergodic since it cycles through only a proper subset of \( \mathbb{R}^2 \).

Since we shall only be concerned with generalizing to the case of using \( \chi_{\zeta} = \zeta \chi_{\zeta} \chi_{\zeta} \ldots \chi_{\zeta} \) on \( \mathbb{R}^N \), we would like to have sufficient conditions that allow us to tell when \( \chi_{\zeta} \) is ergodic. We first need some classical notions from ergodic theory. The following definitions and theorem are found in Halmos [19].

**Definition 2.10.2**

A measure preserving transformation \( \zeta \) is strongly mixing if
\[
\lim_{N \to \infty} P(\xi^{-N} \mathcal{P} \cap \mathcal{Q}) = P(\mathcal{P})P(\mathcal{Q}) \text{ for every pair of measurable sets } \mathcal{P} \text{ and } \mathcal{Q}.
\]

**Definition 2.10.3**

A measure preserving transformation \( \zeta \) is weakly mixing if
\[
\lim_{n \to \infty} \frac{1}{n} \sum_{j=0}^{n-1} |P(\zeta^{-j} \mathcal{P} \cap \mathcal{Q}) - P(\mathcal{P})P(\mathcal{Q})| = 0 \text{ for every pair of measurable sets } \mathcal{P} \text{ and } \mathcal{Q}.
\]

Since ergodicity can be shown to be equivalent to
\[
\lim_{n \to \infty} \frac{1}{n} \sum_{j=0}^{n-1} P(\zeta^{-j} \mathcal{P} \cap \mathcal{Q}) = P(\mathcal{P})P(\mathcal{Q}) \text{ it is easy to show that strong-mixing } \Rightarrow \text{ weak-mixing } \Rightarrow \text{ ergodicity. We have the following result.}
\]

**Theorem 2.10.1 (Halmos)**

A transformation \( \zeta \) on \( \mathbb{R} \) is weakly-mixing iff \( \zeta \chi \zeta \) on \( \mathbb{R}^2 \) is ergodic.

**Proof:**

The proof can be found in Halmos [19], page 39.

The reason that this theorem is useful is now clear. It can readily be shown that multiplicative transformations are strongly-mixing.
and thus, the product machine is an ergodic machine for such generators. For more information on the concepts of mixing and ergodicity, the reader is referred to Billingsley [7] or Halmos [19] where it is also shown that a large variety of transformations are strongly-mixing. This, in fact, turns out to be an easier condition to show than to prove ergodicity.

A generalization of the previous theorem will show that if $\xi$ is a multiplicative generator on $R$, then $\prod_{i=1}^{N} \xi$ is ergodic on $R^N$.

We now consider the problem of faithful transformations in this general setting. In discussing the concept of a faithful transformation, we assumed that the initial value of the seed at time 0 could have been any seed in $R$. Similarly, we shall now assume that when we work with $N$-parallel transformations, each of the $N$ initial seeds is independently chosen and can be any of the seeds in $R$. That is, the initial seed vector in particular is composed of independently chosen seeds from $R$ with the distribution induced by the measure $P$ on $R$.

**Lemma 2.10.1**

Let $E$ be an ergodic machine and let $E$ be the cross-product machine $E \times E$. Then, given the vector sequence $<s(0), s'(0)>, \ldots, <s(n), s'(n)>$, the information loss $IL(s,s')$ on the cross-product space $R^2$ is just $IL(s) \times IL(s')$ where $s$ is the sequence $s(0), \ldots, s(n)$ and similarly for $s'$.

**Proof:**

The proof is by induction. Assume it is true for all vector sequences of length $\leq t$. Then,
\[ \text{IL}(s(t+1), s'(t+1)) = \zeta[\text{IL}(s(t), s'(t)) \cap \{A_s(t), s(t+1) \times A'_s(s(t), s'(t+1)) \} ] \]
\[ = \zeta[\text{IL}(s(t)) \cap A_s(s(t), s(t+1))] \times \{\text{IL}(s'(t)) \cap A'_s(s'(t), s'(t+1)) \} \]
\[ = \zeta[\text{IL}(s(t)) \cap A_s(s(t), s(t+1))] \times \zeta[\text{IL}(s'(t)) \cap A'_s(s'(t), s'(t+1))] \]
\[ = \text{IL}(s(t+1)) \times \text{IL}(s'(t+1)). \]

The proof for the initial case is trivial.

**Theorem 2.10.2**

Suppose \( \zeta \) is a faithful, weakly-mixing transformation for the ergodic machine. Then, \( \prod_{i=1}^{\mathbb{N}} \zeta \) is a faithful transformation on \( E^\mathbb{N} \).

**Proof:**

We first prove the case for \( N = 2 \).

Let \( <s(0), s'(0)>, \ldots, <s(n), s'(n)> \) be any sequence in \( \mathbb{R}^2 \). We must show that the information loss for this sequence is strong. That is, we require that

\[ P[\text{IL}(s,s') \cap \{A_s(n,i \times A'_s(n),j)\}] = P[\text{IL}(s,s')] \cdot P[A_s(n,i \times A'_s(n),j)] \]

for any pseudo state \( <i,j> \).

Now,

\[ P[\text{IL}(s,s') \cap \{A_s(n,i \times A'_s(n),j)\}] \]
\[ = P[\text{IL}(s) \cap A_s(n,i \times \text{IL}(s') \cap A'_s(n),j)] \quad \text{(by previous lemma)} \]
\[ = P[\text{IL}(s) \cap A_s(n,i)] \cdot P[\text{IL}(s') \cap A'_s(n),j] \quad \text{(by definition of probability on the cross-product space)} \]
\[ = P[\text{IL}(s)] \cdot P[A_s(n,i)] \cdot P[\text{IL}(s')] \cdot P[A'_s(n),j] \quad \text{since } \zeta \text{ is faithful} \]
\[ = P[\text{IL}(s,s')] \cdot P[A_s(n,i \times A'_s(n),j)]. \]

Since \( <s,s'> \) was any arbitrary sequence, this proves that the information loss on \( \mathbb{R}^2 \) is always strong and thus, \( \zeta \times \zeta \) is faithful. It is clear that a simple induction argument allows us to generalize this result to the case of a transformation \( \prod_{i=1}^{\mathbb{N}} \zeta \) on \( E^\mathbb{N} \).
The situation for \( N \) seeds operating in parallel with the same generator has thus been fairly easy to handle. This corresponds to having \( N \)-pseudo random number generators working in parallel starting with independent initial seeds. In the case we have been investigating so far, the \( i^{th} \) vector pseudo state component uses the \( i^{th} \) power of the \( i^{th} \) initial seed. However, intuitively, it seems that it should not matter which seed stream one uses, since the next seed element just depends on the generator \( \zeta \) and, in a sense, is completely random. In a case that we actually consider later on, it turns out that we in fact do want to effect a permutation of the seed vector before using it on a particular pseudo state vector.

More formally, let \( \langle r_1, \ldots, r_N \rangle \) be a seed vector. Then, a permutation of this vector is a rearrangement of the seeds. That is, let \( p \) be a bijective mapping from \( \{1, 2, \ldots, N\} \) onto itself. Then, the permutation \( p \) of the seed vector \( \langle r_1, \ldots, r_N \rangle \) will be the new seed vector \( \langle r_{p(1)}, \ldots, r_{p(N)} \rangle \). Although in the actual application of permutations, there will be a particular method as to which permutation is applied at a particular time step, we investigate the more general question of applying any arbitrary permutation at each time step. Since we are allowing arbitrary permutations of the seeds at each time step, the transformation \( \tilde{\zeta} = \prod_{i=1}^{N} \zeta \) does not represent the transition of the seed vector at time \( t \) but rather we first apply the permutation at time step \( t \), that is, apply \( p(t) \), and then apply \( \tilde{\zeta} \) to get the effective transition. That is, we have \( \tilde{\zeta} \circ p(t) \) as the seed space transition at time step \( t \). Now, we are still interested in the fundamental question of information loss for sequences. Even with the notion of permutations, this concept is the same. What we want is transformations that are faithful even when
permutations are applied.

**Definition 2.10.4**

A faithful transformation \( \tilde{\zeta} = \prod_{i=1}^{N} \tilde{\zeta} \) is permutation independent if the information loss function is strong for every vector sequence allowing an arbitrary permutation of the seed vector \( \langle r_1, \ldots, r_N \rangle \) at each time step. We say that \( \zeta \) is permutation independent if for every \( N, \tilde{\zeta} \) is permutation independent.

We now consider the information loss assuming we first have a permutation at time step \( t \).

**Proposition 2.10.1**

Given a \( \tilde{\zeta} = \prod_{i=1}^{N} \tilde{\zeta} \) on \( E^N \)

Let \( p \) be a permutation on \( \{1, \ldots, N\} \) that occurs at time step \( t \), assuming no permutations occurred previously. Let \( s_i(j) \) be the \( i \)th component of the pseudo state at time step \( j \). Then, \( IL(s(0), \ldots, s(t+1)) = \prod_{i=1}^{N} IL(s_p(i)(0), \ldots, s_p(i)(t+1)) \)

**Proof:**

We notice that since \( p \) is a permutation, \( \tilde{\zeta} p = p \tilde{\zeta} \). Thus,

\[
IL(s(0), \ldots, s(t+1)) = \tilde{\zeta} p IL(s(0), \ldots, s(t)) \cap A(s(t), s(t+1)) = p \tilde{\zeta} \prod_{i=1}^{N} [IL(s_i(0), \ldots, s_i(t)) \cap A(s_i(t), s_i(t+1))] = p \prod_{i=1}^{N} [IL(s_i(0), \ldots, s_i(t+1))] = \prod_{i=1}^{N} IL(s_p(i)(0), \ldots, s_p(i)(t+1))
\]

We see that the information loss is simply a rearrangement of
the cross-product terms in the information loss function without permutations. This observation will give us sufficient conditions under which a transformation can be permutation independent.

**Proposition 2.10.2**

For the information loss to be strong at time step \( t+1 \), given a permutation \( p \) at time step \( t \) and not before, the following must hold.

\[
P[IL(s_p(i)(0), \ldots, s_p(i)(t+1)) \cap A_{s_i(t+1), j}] = P[IL(s_p(i)(0), \ldots, s_p(i)(t+1))] \cdot P[A_{s_i(t+1), j}]
\]

for all pseudo states \( j \).

**Proof:**

The proof of this is simply the definition of the information loss being strong and the previous proposition.

Notice that in the previous proposition, the intersection is with the nonpermuted \( i^{th} \) pseudo-state component. Thus, the intersection of the information loss must be strong with every transition from state \( i \) to state \( j \) in general. For our purposes, these two propositions give us the sufficient conditions we need, for we get the following theorem.

**Theorem 2.10.3**

If the information loss is \( R^N \) at every time step for a faithful transformation \( \chi_{i=1}^N \zeta \) on \( E^N \), then \( \zeta \) is permutation independent.

**Proof:**

By proposition 2.10.2 and the fact that each component of the information loss is \( R \) at time step \( t \), we see that the information loss is \( R \) at time step \( t+1 \) for each component. Thus, we are effectively starting at time step 0 after the permutation. Consequently, for
permutations at each time step, we still have the information loss $R$ for each component at every time step and thus it is strong.

We see that for multiplicative generators, if they are faithful with the information loss $R$ at each time step, then permutation does not alter the information loss. We conclude this chapter with remarks about considering ergodic machines that are not autonomous.

2.11 Nonautonomous Ergodic Machines

Most of our results up to now have been derived for the case of autonomous ergodic machines. The definitions of information loss and gain and faithfulness of a transformation $\xi$ have also been geared to autonomous machines. However, this was for the sake of understandability and convenience of notion only. It is clear that the generalization to nonautonomous ergodic machines is quite easy. For the case of autonomous ergodic machines, the seed transition space was the set $A_{ij}$ from state $i$ to state $j$. For the general case, this seed space must be defined for each $x \in X$ where $X$ is the input space. Thus, given the input $x$, there is a set $A_{ij}(x) \in \mathbb{R}$ such that if the ergodic machine is in pseudo state $i$ then the next pseudo state is $j$. Clearly, we can still consider the concept of information loss and information gain except now, we shall have to consider the loss and gain of a sequence $s(0), \ldots, s(n)$ as also indexed by the input sequence $x(0), \ldots, x(n)$. Similarly, we would have to generalize the concept of a sequence being strong to include every possible input at the next time step. For example, the following would be the definition of the forward information loss.

Definition 2.11.1

The FIL of a sequence $s(0), s(1), \ldots, s(n)$ with input $x(0), \ldots, x(n)$
is defined recursively as follows.

(1) \( \text{FIL}(0) = R \)

(2) \( \text{FIL}(k) = \xi[\text{FIL}(k-1) \cap A_{s(k-1),s(k)}(x(k-1))] \)

Similarly, we would define the concept of the information loss being strong as:

**Definition 2.11.2**

Given the pseudo state sequence \( s(0), \ldots, s(n) \) and the input sequence \( x(0), \ldots, x(n) \), the information loss is **strong** at time \( n \) for this paired sequence if for all \( j \in S \)

\[
P[\text{IL}(s,x) \cap A_{s(n),j}(x(n))] = P[\text{IL}(s,x)] \cdot P[A_{s(n),j}(x(n))] \]

The rest of the definitions are essentially based on these concepts and generalize easily. Care must be taken however in realizing that we are considering \( X \) to be an arbitrary set in general. Thus, when we take the minimum measure of the transition seed spaces \( \min_{i,j,x} P[A_{i,j}(x)] \)

we must realize that we are now minimizing over a possibly uncountable number of seed spaces. Clearly, the faithfulness of generators will depend on how complicated the resulting seed spaces \( A_{i,j}(x) \) are allowed to be.

Keeping in mind these considerations, the generalization to arbitrary ergodic machines is thus straightforward.

**2.12 Conclusion**

The theoretical investigations of this chapter have provided us with enough machinery to aptly represent probabilistic models in a deterministic framework. The development has shown that faithful representative models exist. We have also shown that one should be careful in trying to use the same generator for different ergodic machines. We
shall return to the theoretical development in Chapter IV where we introduce the concept of a probabilistic homomorphism. In the next chapter, however, we experimentally investigate a neuron network model. It will be apparent that the model can in fact be represented in ergodic machine terms. We now turn to this neuron network model which is essentially an investigation of how an approximate homomorphism can preserve information.
CHAPTER III

APPROXIMATE HOMOMORPHISMS--SIMPLIFICATIONS

IN A NETWORK OF NEURONS

In this chapter, we study the simplification of a network of interconnected neurons. The idealized neuron, which will be the atomic unit or component of the network, can be formally represented by a probabilistic automaton and thus, by an ergodic machine. The network of neurons will be a base model, and we investigate the relationship of this model with that of a specific lumped model that we shall consider. The atomic unit of the lumped model will be called a block, and will represent an aggregation of similar neurons in the base model. These neurons that form the block will be required to satisfy certain regularity conditions in their connection pattern. Also, each neuron in a block will have exactly the same parameters, that is, will be represented by the same ergodic machine, although the neurons will have varying behavior since they will be perturbed independently by "random" noise. The number of neurons in a block and the number of neighbors per neuron will be a measure of the "size" of the block. Although this notion can be formalized, we shall be content with observing that a block with more neurons and more neighbors per neuron has a greater size than one with fewer neurons and fewer neighbors per neuron.

In actuality then, rather than considering a single base model, we are really considering a class of base models. Two base models can
differ either by having different parameters for their components, or by being of different sizes. It will be the case that we shall use the same lumped model for base models that differ only in size.

Although we are carrying out a detailed investigation of a network of idealized neurons, the fact that they represent neurons is not fundamental. We are actually studying a method for reducing complexity in a network of ergodic machines. For example, in studying an ecological system, one might model an individual by an ergodic machine and consider a species to be the aggregation of similar units. Thus, a block would represent a species. Similar translations can be undertaken in analyzing networks in other disciplines. However, for simplicity, we shall use neurons as our continuing example.

It is useful to explicitly make some comments on the type of simplification that we are considering. We are considering aggregations of identical components (e.g. neurons) into blocks whose output we wish to preserve under the lumping. For each block, the output will be the percentage of neurons in the specific neuron states. This viewpoint influences what we consider important behavior. Thus, there may be micro-oscillations in the same block of the base model that we would not be aware of if the percentage vector for that block remained the same. For example, suppose two neurons in the same block simply alternated states. This would be a micro-oscillation that we would not be aware of since the percentage vector for that block could not discriminate for such behavior. On the other hand, suppose each block represented a cell assembly, and activity of an assembly corresponded to percent activity of the block. Then, we could recognize oscillations between two such blocks. We consider such an example later on. Whitehead [38] for ex-
ample, is studying groups of cell assemblies represented by blocks where an assembly (block) is dominant over another if the percentage firing of that assembly is greater than another assembly. We now turn to the concept of an approximate homomorphism.

3.1 Approximate Homomorphisms

The base model-lumped model pair that we investigate consists of a probabilistic base model and a deterministic lumped model. The lumped model is, in fact, derived from a consideration of the limiting case of the base model. We take the idealized case of a base model that has a countably infinite number of neurons and a countably infinite number of neuron neighbors per neuron. This base model, call it \( B_n \), can be thought of as the limit as the size becomes infinite of models \( B_{n_i} \) where \( n_i \) indexes a particular base model of a specific size, that is, \( B = \lim_{n} B_n \).

In the next chapter, we show that the lumped model, call it \( L \), is a homomorphic image of the model \( B \). Actually, since the base model is probabilistic, we introduce the concept of a p-homomorphism between two ergodic machines. This allows us to say that the homomorphism relation exists between \( B \) and \( L \). However, in this chapter, we will only investigate the relationship between \( B_{n_i} \) and \( L \) for various values of \( n \). It turns out that much information is still preserved and that analyzing \( L \) gives us a reasonable idea of the behavior of \( B_{n_i} \). We shall call the relationship between \( B_{n_i} \) and \( L \) an approximate homomorphism. That is, \( L \) is an approximate homomorphic image of \( B_{n_i} \) if there exists \( B \) such that \( \lim_{n} B_n = B \) and \( L \) is a homomorphic image of \( B \). We shall not bother to be more formal about the concept of approximate homomorphism since we do not use it formally in any manner.
3.2 The Neuron Model

The base model-lumped model pair that we give a detailed description of in this section was first considered by Zeigler [44]. Both lumped and base models operate on a discrete time scale. This is natural in our context since we are studying simplification in the context of ergodic machines which also operate on a discrete time scale. Since the time scale can be made arbitrarily fine, this is really no restriction in modeling such processes as neuron behavior. Furthermore, most actual models of neural nets actually are modeled on a discrete time scale.

3.2.1 The Base Model

Components

NEURONS--elements modeling basic behavior assumed to be characteristic of real brain cells.

INPUT WIRES--sources of external net excitation.

Descriptive Variables

For each NEURON:

RECOVERY STATE--with range the non-negative integers

RECOVERY STATE = i indicates that i time units have elapsed since the NEURON last fired; thus i=0 means the NEURON is now firing.

NOISE--with range the real numbers; (when NOISE = r the actual threshold for firing of the NEURON is THRESHOLD (i) + r where THRESHOLD (i) is the threshold value characteristic of RECOVERY STATE i. NOISE is a random variable generated for each NEURON.

STRENGTH--with range the real numbers; STRENGTH = x indicates that the sum total of all NEURON inputs is x.
FIRING\_STATE--with range \{0,1\}; 1 means the NEURON is firing (emitting a pulse), 0 that it is not firing.

For each INPUT\_WIRE

INPUT\_LEVEL--with range the real numbers;

INPUT\_LEVEL = x indicates that the excitation level on the INPUT\_WIRE is x (measured in the same units as STRENGTH).

For each BLOCK (designated set of NEURONS)

and for each RECOVERY\_STATE\_i:

\% IN\_STATE\_i--with range the rationals in [0,100]; the percentage of NEURONS in the BLOCK in RECOVERY\_STATE\_i.

PARAMETERS:

For each BLOCK:

NOISE\_DISTRIBUTION--the probability distribution for random variable NOISE.

THRESHOLD--a real valued function with RECOVERY\_STATE as argument, giving the minimum strength needed to fire a NEURON in the absence of NOISE.

For each NEURON:

NEURON\_NEIGHBORS--with range the subsets of NEURONS indicates the NEURONS whose outputs impinge on the NEURON.

EXTERNAL\_NEIGHBORS--with range the subsets of INPUT\_WIRES indicates the external inputs impinging on the NEURON.

and for each NEIGHBOR (NEURON or EXTERNAL) of the NEURON:

SYNAPS--with range the reals; determines the influence that a NEIGHBOR has on the NEURON: positive indicates
facillatory, negative indicates inhibitory.

**Component Interaction**

Postulate 1. Each NEURON receives inputs from other NEURONS called NEURON-NEIGHBORS as well as from external sources called EXTERNAL-NEIGHBORS. We just refer to NEIGHBORS when the context is clear.

Postulate 2. The NEURONS are grouped into disjoint sets called BLOCKS. This partitioning of the NEURONS places the following restrictions on the possible NEURON-NEIGHBORS of a NEURON:

Suppose that a NEURON α belongs to BLOCK i.

a) If α has some NEIGHBORS in BLOCK j (which may equal i) then every NEURON in BLOCK i must have at least one NEIGHBOR in BLOCK j.

b) The SYNAPS weight is the same for the outputs of all NEIGHBORS of α from the same BLOCK. The NEIGHBORS of α coming from BLOCK j all have the same "influence" on it, but this may be different from that of the NEIGHBORS of α coming from BLOCK k (k≠j).

c) BLOCK j exerts the same "influence" on each NEURON in BLOCK i. We measure the "influence" of BLOCK j on α by the product of the number of α's NEIGHBORS from BLOCK j and their common SYNAPS value. This product is to take the same value for all NEURONS in BLOCK i, so that the "influence" of BLOCK j on each NEURON in BLOCK i is the same.

d) The characteristics of all NEURONS in a BLOCK, namely the THRESHOLD function and NOISE probability
distribution, are the same.

e) The EXTERNAL·NEIGHBORS and their SYNAPS weights are the same for all NEURONS in a BLOCK.

Postulate 3. Each NEURON operates at each time step as follows:

a) The STRENGTH of the input volley is computed by adding together the outputs (which may be 0 or 1) of all the EXTERNAL·NEIGHBORS and NEURON·NEIGHBORS of a weighted by their respective SYNAPS values.

b) If this STRENGTH is great enough, the NEURON will fire, i.e. will put out a 1 for the next time step; otherwise it will put out a 0. The 1/0 output can be thought of as a pulse/no pulse sent out by the NEURON, with only the pulse being able to influence the activities of another NEURON.

c) The minimum STRENGTH required to fire is dependent on the RECOVERY·STATE of the NEURON and NOISE. Generally, the longer it has been since the NEURON last fired, the easier it is to fire, indeed if it has just fired it may be impossible to fire for some time (this period is called the absolute refractory period). This fact is formalized in the shape of the THRESHOLD function. This function determines for each RECOVERY·STATE value i (time since last fired) a value, THRESHOLD (i) if there were no noise.

It is also assumed that the NEURON is perturbed by additive noise independently generated at each step for each NEURON from the NOISE probability distribution.
Thus if the NEURON is now in RECOVERY·STATE i, 
the minimum STRENGTH required to fire it is THRESHOLD 
(i) plus the sample value of NOISE. 
d) If the NEURON fires, its RECOVERY·STATE is set to 
0, otherwise it is incremented by 1. Thus, as implied, 
it records the time elapsed since the last firing of 
this NEURON.

This completes the description of the base model. It can easily 
be seen that the basic component, the neuron, can be represented formally 
as an ergodic machine.

3.2.2. The Lumped Model

Components

BLOCK'S--each BLOCK' represents in lumped form the behavior of 
a corresponding BLOCK of base model NEURONS. (' indicates the correspondence.)

INPUT·WIRES's--each INPUT·WIRE' of the lumped model represents 
a corresponding INPUT·WIRE of the base model.

Descriptive Variables

For each BLOCK':

For each i = 0,1,2,....

%·IN·STATE·i'--with range [0,100]; %·IN·STATE·i' = q 
predicts that q/100 of the base model NEURONS in BLOCK 
are in RECOVERY·STATE·i. Thus %·IN·STATE·i' of BLOCK' 
corresponds to %·IN·STATE·i of BLOCK. We use the term 
PROBABILITY·IN·STATE·i interchangeably with %·i·STATE·i'.

STRENGTH'--with range the real numbers; STRENGTH' = x'
indicates that the sum total of inputs to BLOCK' is \( x' \).

For each INPUT·WIRE':

INPUT·LEVEL'--with range the real numbers.

PARAMETERS:

For each BLOCK':

BLOCK'·NEIGHBORS--with range the subsets of BLOCK's; specifies the BLOCK's which send input to the BLOCK'.

EXTERNAL'·NEIGHBORS--with range the subsets of INPUT·WIRES'.

and for each NEIGHBOR (BLOCK' or EXTERNAL'):

WEIGHT'--with range the real numbers; determines the influence of the NEIGHBOR on the BLOCK'.

NOISE·DISTRIBUTION'--a cumulative density function (c.d.f.)

(will be the c.d.f. of NOISE).

THRESHOLD'--a function from the non-negative integers to the reals (to be identical with THRESHOLD).

Component Interaction

1. Each BLOCK' receives input from other BLOCK's (called BLOCK'·NEIGHBORS) and external sources (called EXTERNAL'·NEIGHBORS).

2. Each BLOCK' at each time step operates as follows:
   a) The STRENGTH' of the input to the BLOCK' is computed as the weighted sum of all NEIGHBOR (BLOCK' and EXTERNAL')
      outputs.
   b) Using this STRENGTH' and the current values of the \%·IN·STATE·i' (one for each RECOVERY·STATE i) the next values
      of the \%·IN·STATE·i' variables are computed. For each i, the current value of \%·IN·STATE·i' is multiplied by
      \( 1 - F(x - \text{THRESHOLD'}(i)) \) to obtain the new value of \%·IN·STATE·i+1'.
Here, \( F \) is the \text{NOISE\cdot DISTRIBUTION}' (i.e., the cumulative distribution function), and \( x \) is the value of \text{STRENGTH}'. This accounts for the fraction of base model \text{NEURONS} in \text{RECOVERY\cdot STATE}\cdot i \) which have not fired (hence go to \( i+1 \)); the remainder then, are those that do fire (hence go to \text{RECOVERY\cdot STATE}\cdot 0) and the lumped model reflects this as a sum of all such contributions to obtain the new \( \%\cdot \text{IN\cdot STATE}\cdot 0' \). We will discuss the derivation of this lumped transition rule after we define the correspondence between the base and lumped structures.

c) The output of \text{BLOCK}' is just \( \%\cdot \text{IN\cdot STATE}\cdot 0' \), which, if the lumped model is valid, is the percentage of \text{NEURONS} in the \text{BLOCK} which are firing, i.e., outputting a pulse \( (1) \). (The rest are in the various \text{RECOVERY\cdot STATES} and since they output a 0, will not influence any activity at the next time step.)

\textit{Correspondences between lumped and base model}

\textit{Components}

Each \text{BLOCK}! (lumped model component) corresponds to a \text{BLOCK} (partition class) of \text{NEURONS} (base model components).

\textit{Descriptive Variables}

For each \text{BLOCK}!:

For each \( i = 0,1,2,... \)

\( \%\cdot \text{IN\cdot STATE}\cdot i' = \%\cdot \text{IN\cdot STATE}\cdot i \)

\( \text{STRENGTH}' = \text{STRENGTH} \) of any \text{NEURON} in the corresponding \text{BLOCK} (this \text{STRENGTH} is the same for each \text{NEURON} in a \text{BLOCK} by base model postulate 2).
PARAMETERS:

INPUT'\cdot WIRES = INPUT\cdot WIRES

for each BLOCK':

BLOCK'\cdot NEIGHBORS = the set of BLOCK's whose corresponding
BLOCKs influence NEURONS in the BLOCK (corresponding to
BLOCK').

EXTERNAL'\cdot NEIGHBORS = EXTERNAL\cdot NEIGHBORS of any NEURON in the
the BLOCK (a BLOCK constant according to postulate 2).

THRESHOLD' = THRESHOLD

NOISE\cdot DISTRIBUTION' = NOISE\cdot DISTRIBUTION for any NEURON
in the BLOCK (postulate 2, again).

and for each NEIGHBOR (BLOCK' or EXTERNAL')

WEIGHT' = the product of the number of corresponding
NEIGHBORS and their common SYNAPS values (postulate 2)
of any NEURON in BLOCK.

for each INPUT\cdot WIRE':

INPUT\cdot LEVEL' = INPUT\cdot LEVEL

Component Interaction Continued

We are now interested in deriving the lumped model transition
function. The state of the lumped model is described by a vector
$\%IN\cdot STATE'\cdot i$ for each BLOCK'. We wish to calculate how this vector
changes at each time step. Suppose at time step $t$, this state vector
for a specific BLOCK' is represented by $s(t) = <s_1(t), s_2(t), \ldots, s_n(t)>$
where $s_i(t)$ represents the $\%IN\cdot STATE'\cdot i$. We are assuming that there
are $n$ recovery states.

Consider the information that we have for each NEURON in BLOCK
corresponding to BLOCK'. We know that it is perturbed independently by noise at each time step and that its threshold is a function of the recovery state. Also, for this BLOCK, the THRESHOLD function and NOISE DISTRIBUTION function are the same for each NEURON in the BLOCK and the same as for BLOCK'. Consequently, we can compute \( p_{i,i+1}(x(t)) \), the probability that the NEURON goes next to RECOVERY·STATE·i+1 (does not fire) given its present RECOVERY·STATE·i and the input STRENGTH = \( x(t) \). Assume that \( F(z) \) is the cumulative noise distribution function for the BLOCK under consideration. We also assume that a neuron that does not fire in recovery state \( n \) remains in state \( n \) and thus \( p_{n,n+1}(x(t)) = p_{n,n}(x(t)) \) is the probability that a neuron in state \( n \) remains in \( n \). We thus have,

\[
\begin{align*}
  p_{i,i+1}(x(t)) &= \text{Prob}\{x(t) < \text{THRESHOLD}(i) + z\} \\
  &= 1 - F(x(t) - \text{THRESHOLD}(i))
\end{align*}
\]

Also, since a NEURON can only go to RECOVERY·STATE·0 if it does not go to \( i+1 \), we have

\[
p_{i,0} = 1 - p_{i,i+1}(x(t)) = F(x(t) - \text{THRESHOLD}(i))
\]

We can present these conditional probabilities in the form of a matrix where each \( p_{ij} = p_{ij}(x(t)) \)

\[
P(x(t)) = \begin{bmatrix}
  p_{00} & p_{01} & 0 & \ldots & 0 \\
  p_{10} & 0 & p_{12} & \ldots & 0 \\
  \vdots & \vdots & 0 & \ddots & \vdots \\
  \vdots & \vdots & \vdots & \ddots & p_{n-1,n} \\
  p_{n0} & 0 & 0 & \ldots & p_{nn}
\end{bmatrix}
\]

Given the matrix \( P(x(t)) \) and the \(<\%·IN·STATE'·i>\) (row) vector \( s(t) \) we shall define the transition behavior as:

\[
s(t+1) = s(t) P(x(t))
\]
It turns out that if we let this vector \( s(t) = \langle \%\text{IN\_STATE}\cdot i \rangle \) be the corresponding vector to the \( \langle \%\text{IN\_STATE}\cdot i \rangle \) of the corresponding BLOCK in the base model, then in a model \( B \) that is the idealized version, with an infinite number of neurons and an infinite number of neighbors per neuron, the correspondence is preserved over time using the above transition matrix. We show this formally in the next chapter. Intuitively, however, the reason it works is that since each neuron has an infinite number of neighbors, we can assume that each neuron receives the same identical input. Furthermore, the fact that there are an infinite number of neurons in a block will say that the correct percent will not fire in any \( \text{RECOVERY\_STATE}\cdot i \), given the noise distribution.

In the rest of this chapter, we experimentally investigate the correspondence between realizable base models \( B_n \) and their approximate homomorphic image \( L \). We shall find it convenient to consider the above matrix a Markov matrix for purposes of analysis since the entries can be thought of as transition probabilities. If the strength \( x(t) \) is a constant at each time step, then the matrix can be thought of as a homogeneous Markov matrix. This conceptualization of the matrix as a homogeneous or inhomogeneous matrix is for the purpose of using Markov theory to study the limiting forms of the \( \langle \%\text{IN\_STATE}\cdot i \rangle \) vector only, since the meaning of \( \%\text{IN\_STATE}\cdot i \) is not the probability of being in a particular recovery state for the BLOCK. The lumped model is in fact deterministic and there are no probabilities associated with it.

### 3.3 The Computer Simulation

In order to investigate the approximate homomorphism between the base and lumped model, we carried out a computer simulation of the two models. The simulation consisted of a package of Fortran programs that
allowed great flexibility in specifying connection patterns, initial states and parameter settings. Certain statistics were automatically compiled. A pseudo random number generator was used to create a stream of "random" numbers to be used for NOISE. Further information on the simulation programs can be found in Appendix II.

Two restrictions on the base model were imposed in order to simplify the computer implementation.

(1) There is a maximum of 20 recovery states possible. A neuron in the highest recovery state that does not fire remains in the same recovery state. In our test runs we only use recovery states 0-6.

(2) For all i, j, all neurons in BLOCK i have the same number of neighbors in BLOCK j with equal synapse weights and these neighbors are randomly chosen from the neurons in BLOCK j.

In the following sections, we will consistently use "run" to mean a single simulation of the base or lumped model with a fixed set of parameters and initial random number seed. An "experiment" will refer to several runs with identical parameters but different random number seeds. A set of related experiments will be called a "series of experiments" or simply a "series".

Since the EXTERNAL-NEIGHBORS and their SYNAPS weights are the same for all neurons in a BLOCK, we will use the term "external input" for the external influence.

The next three sections are three different series of experiments. For the noise in each neuron, we always used a Gaussian distribution. There seemed to be no reason to consider other noise distributions although the computer implementation allowed them. Also, to simulate an absolute refractory period and a relative refractory period, it was
found convenient to use an exponential decay function for the threshold, although again, an arbitrary threshold function could have been used. It was sometimes convenient to always have a fixed background level of input to each neuron and this is reflected in the parameter setting background. This can either be considered as part of the external input or as a modification to the threshold.

The first series of experiments simply investigates the situation where there are no connections but just aggregations of neurons each of which works autonomously.

3.4 Single Block--No Connections

We first describe the basic structure of this series of experiments.

Connection Pattern

The base model consisted of a single block with no connections between neurons in the block. We varied the size of the block using 25, 50, 100, 500 and 1000 neurons.

Parameter Settings

1. Background: 0
2. Noise: Gaussian, with mean 0 and standard deviation of 10, 20 and 40.
3. Threshold: Exponential decay $27e^{-s}$ where s is the recovery state.

Initial State

All neurons were initially in recovery state 6.

Input Segment

Input levels of -100, -80, ..., 80, 100 were tried. The length of the input segment was sufficient to allow a "steady state" to be
reached.

Not all possible combinations of experiments were conducted. The following shows the experiments actually done (x's).

<table>
<thead>
<tr>
<th>Size</th>
<th>St. Dev.</th>
<th>25</th>
<th>50</th>
<th>75</th>
<th>100</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Since there are no connections, the only input that a neuron receives is external input and consequently, each neuron receives the same total input. The only difference is that each neuron samples independently from a noise distribution. Let us assume that there is an exact agreement between the percent-in-state vector of recovery states in the lumped model and the percent-in-state vector of the base model at time t. Then, at time t+1, the actual percent-in-state vector of the base model can be considered to be simply a sample from the distribution predicted by the lumped model.

Whether or not the observed differences between the actual percent-in-state vectors and the predicted percent-in-state vector are reasonable can be tested using the chi-square statistic. In effect, we would be testing the random number generator. Such a chi-square test would surely have given good results if the lumped model was reset at each time step. However, the aim of our investigations is to show that the lumped model can be used to estimate behavior of the base model without of course knowing the base model behavior. Resetting the lumped model correctly would clearly be possible only if base model behavior is known. Without resetting the lumped model, error propagation becomes an
added consideration and is discussed later.

Before giving the results of the chi-square tests, it will be instructive to examine the behavior of the lumped model. Because there are no connections between blocks, for a fixed level of external input the lumped model can be analyzed as if it were a homogeneous Markov chain with the following transition matrix:

$$
P = \begin{bmatrix}
p_{00} & 1-p_{00} & 0 & 0 & 0 & 0 & 0 \\
p_{10} & 0 & 1-p_{10} & 0 & 0 & 0 & 0 \\
p_{20} & 0 & 0 & 1-p_{20} & 0 & 0 & 0 \\
p_{30} & 0 & 0 & 0 & 1-p_{30} & 0 & 0 \\
p_{40} & 0 & 0 & 0 & 0 & 1-p_{40} & 0 \\
p_{50} & 0 & 0 & 0 & 0 & 0 & 1-p_{50} \\
p_{60} & 0 & 0 & 0 & 0 & 0 & 1-p_{60}
\end{bmatrix}
$$

The actual values of the entries can be easily calculated. For a fixed level of external input $x$ (note that this is the same at each time step $t$), if $F$ is the cumulative distribution function (c.d.f.) that is Gaussian with mean 0 and standard deviation $\sigma$, $(N(0,\sigma))$, we have

$$p_{10} = F(x - 27e^{-t})$$

Using the terminology of Feller [15], we see that viewed as a Markov Chain, the transition matrix is irreducible, aperiodic and positive recurrent. Consequently, it possesses a long run distribution. That is, regardless of the initial % in state vector of recovery states, the lumped model will approach the same steady state distribution in the limit. The limiting distributions for various input levels are shown in figs. 3.4.1 and 3.4.2.
Figure 3.4.1. Lumped Model Steady State Distribution \( \sigma=10 \)
The behavior of the base model was qualitatively similar to that of the lumped model although the base model percentages tended to fluctuate about a "steady state." A quantitative analysis was obtained using the standard chi-square test. For each time step we calculated a chi-square value assuming that the base model percentages were a sample from the theoretical distribution represented by the lumped model % in state vector. If the expected number of neurons in a particular recovery state was less than five, then this recovery state was aggregated with another recovery state. Thus, the number of degrees of freedom ranged from 0 to 6. For a given number of degrees of freedom R, the chi-square statistics obtained should be $\chi^2_R$ distributed. Tables 3.4.1 and 3.4.2 show the distributions that were actually obtained for each experimental run. The values do in fact appear to be $\chi^2_R$ distributed. A second level chi-square analysis was conducted assuming the hypothesis that we were sampling from a chi-square distribution. Table 3.4.3 shows the results of this analysis.

One would not expect the chi-square tests between the lumped and base models to be extremely good because of the problem of error propagation. The chi-square test assumes that sampling is independent from time step to time step. Thus, if at each time step we were to reset the lumped model, we would expect good chi-square results. One way to eliminate most of the correlation from one time step to the next due to not resetting the lumped model is not to sample at every time step. Rather, one might sample say at every fifth time step. We did an analysis sampling at every fifth time step for the case of 100 neurons with a standard deviation of 20. The results obtained were extremely good and are also shown in tables 3.4.1 and 3.4.3.
<table>
<thead>
<tr>
<th>Size</th>
<th>St. Dev.</th>
<th>Sample Size</th>
<th>Deg. Freedom</th>
<th>Distribution of Chi-Square Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>20</td>
<td>93</td>
<td>2</td>
<td>99, 96, 88, 74, 60, 8</td>
</tr>
<tr>
<td>50</td>
<td>20</td>
<td>30</td>
<td>2</td>
<td>100, 97, 90, 73, 37, 3</td>
</tr>
<tr>
<td>75</td>
<td>20</td>
<td>30</td>
<td>2</td>
<td>100, 97, 97, 80, 47, 3</td>
</tr>
<tr>
<td>100</td>
<td>20</td>
<td>44</td>
<td>2</td>
<td>98, 95, 91, 73, 36, 7</td>
</tr>
<tr>
<td>500</td>
<td>20</td>
<td>42</td>
<td>2</td>
<td>95, 93, 88, 74, 62, 2</td>
</tr>
<tr>
<td>1000</td>
<td>20</td>
<td>42</td>
<td>2</td>
<td>100, 98, 90, 83, 48, 5</td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>12</td>
<td>2</td>
<td>100, 100, 92, 83, 58, 8</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>36</td>
<td>2</td>
<td>100, 97, 94, 80, 50, 3</td>
</tr>
<tr>
<td>25</td>
<td>40</td>
<td>96</td>
<td>2</td>
<td>100, 95, 90, 79, 49, 4</td>
</tr>
<tr>
<td>100</td>
<td>20</td>
<td>174</td>
<td>2</td>
<td>99, 95, 91, 74, 48, 6</td>
</tr>
</tbody>
</table>

(last row uses samples from every 5th time step)

The Theoretical distribution is: 99, 95, 90, 75, 50, 5
TABLE 3.4.2

$\chi^2$ STATISTICS OF BASE MODEL VS. LUMPED MODEL

(PART 2)

<table>
<thead>
<tr>
<th>Size</th>
<th>St. Dev.</th>
<th>Sample Size</th>
<th>Deg. Freedom</th>
<th>Distribution of Chi-Square Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>20</td>
<td>66</td>
<td>1</td>
<td>100 97 88 73 53 0</td>
</tr>
<tr>
<td>50</td>
<td>20</td>
<td>66</td>
<td>4</td>
<td>100 97 92 79 48 0</td>
</tr>
<tr>
<td>75</td>
<td>20</td>
<td>51</td>
<td>4</td>
<td>100 100 100 84 61 10</td>
</tr>
<tr>
<td>100</td>
<td>20</td>
<td>54</td>
<td>5</td>
<td>96 94 90 68 39 7</td>
</tr>
<tr>
<td>500</td>
<td>20</td>
<td>105</td>
<td>6</td>
<td>100 97 93 76 40 1</td>
</tr>
<tr>
<td>1000</td>
<td>20</td>
<td>105</td>
<td>6</td>
<td>98 94 88 73 57 7</td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>57</td>
<td>3</td>
<td>100 100 95 82 60 16</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>48</td>
<td>5</td>
<td>98 96 90 77 54 4</td>
</tr>
<tr>
<td>25</td>
<td>40</td>
<td>32</td>
<td>3</td>
<td>100 94 78 78 59 3</td>
</tr>
</tbody>
</table>

The theoretical distribution is: 99 95 90 75 50 5
<table>
<thead>
<tr>
<th>Size</th>
<th>St. Dev.</th>
<th>Deg. Freedom</th>
<th>Sample Size</th>
<th>2nd Level Chi-Square</th>
<th>Deg. of Freedom</th>
<th>% Confidence of Acceptance</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>20</td>
<td>2</td>
<td>93</td>
<td>8.27</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>50</td>
<td>20</td>
<td>2</td>
<td>30</td>
<td>2.65</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>75</td>
<td>20</td>
<td>2</td>
<td>30</td>
<td>2.30</td>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>100</td>
<td>20</td>
<td>2</td>
<td>44</td>
<td>4.93</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>500</td>
<td>20</td>
<td>2</td>
<td>42</td>
<td>4.88</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>1000</td>
<td>20</td>
<td>2</td>
<td>42</td>
<td>3.72</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>2</td>
<td>12</td>
<td>0.17</td>
<td>2</td>
<td>90</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>2</td>
<td>36</td>
<td>1.58</td>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>25</td>
<td>40</td>
<td>2</td>
<td>96</td>
<td>2.54</td>
<td>5</td>
<td>75</td>
</tr>
<tr>
<td>25</td>
<td>20</td>
<td>1</td>
<td>66</td>
<td>2.05</td>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>50</td>
<td>20</td>
<td>4</td>
<td>66</td>
<td>3.43</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>75</td>
<td>20</td>
<td>4</td>
<td>51</td>
<td>1.38</td>
<td>3</td>
<td>70</td>
</tr>
<tr>
<td>100</td>
<td>20</td>
<td>5</td>
<td>54</td>
<td>4.63</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>500</td>
<td>20</td>
<td>6</td>
<td>105</td>
<td>7.67</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>1000</td>
<td>20</td>
<td>6</td>
<td>105</td>
<td>5.27</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>3</td>
<td>57</td>
<td>1.80</td>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>5</td>
<td>48</td>
<td>0.56</td>
<td>3</td>
<td>90</td>
</tr>
<tr>
<td>25</td>
<td>40</td>
<td>3</td>
<td>32</td>
<td>8.33</td>
<td>3</td>
<td>2.5</td>
</tr>
<tr>
<td>25</td>
<td>20</td>
<td>2</td>
<td>174</td>
<td>5.27</td>
<td>5</td>
<td>90</td>
</tr>
</tbody>
</table>

(Last row uses samples from every 5th time step)
We shall now consider the question of error propagation. Zeigler [45] introduced the concept of considering the error at a time step as composed of two parts: (1) due to sampling error and (2) due to not resetting the lumped model. The second component of the error can be analyzed solely in terms of lumped model behavior. Fig. 3.4.3 is a schematic representation of error propagation. Because of the structure of the approximate homomorphism $h$, $\epsilon_1$ and $\epsilon_2$ are due to sampling errors. The additional error $k\epsilon_1$ is due to not resetting the lumped model. To investigate this additional error, we will describe what happens to two states of the lumped model that differ by a distance $\varepsilon$.

Assume that $r$ and $s$ are two different percent in state vectors (states) of the lumped model. That is, $r = <r_0, \ldots, r_6>$ and $s = <s_0, \ldots, s_6>$. Let distance be measured by the metric $d$ defined as:

$$d<r, s> = \max_{0\leq i \leq 6} |r_i - s_i|$$

Let $a = r - s$. Then, $a$ will be called the error vector. Notice that

$$\sum_{i=0}^{6} a_i = 0.$$ We have

$$d<rP, sP> = \max_{i} |(aP)_i|$$

where

$$aP = \left( \sum_{i=0}^{6} a_i p_{i0}, a_0(1-p_{00}), a_1(1-p_{10}), \ldots, a_4(1-p_{40}), a_5(1-p_{50}), a_6(1-p_{60}) \right).$$

Now, if there exists $k < 1$, such that for all pairs of lumped model states $r, s, dr, s > \leq kd<r, s>$

then the error propagation can be shown to be ultimately bounded.

(Zeigler [45]) Unfortunately, a simple example shows that such a condition cannot be guaranteed. Consider the two percent in state
Figure 3.4.3. Error Propagation
vectors

\[ r = <.8,0,0,0,0,.1,.1> \text{ and } s = <.9,.1,0,0,0,0,0>. \]

Let the external input be -20 and the standard deviation of the noise be 20. The \( p_{00} = .01, p_{10} = .07, p_{20} = .12, p_{30} = .14, p_{40} = .15, p_{50} = .16, p_{60} = .16. \)

\[ a = <-.1,-.1,0,0,0,.1,.1> \]

\[ aP = <.024,-.099,-.093,0,0,0,.168> \]

Thus, \( d<r,s> = .1 \) but \( d<rP,sP> = .168. \)

We see that \( k \) must be greater than 1. In one transition then, we cannot be assured that two lumped model states will converge. In this case, bounded error propagation does not apply.

An interesting situation is the case of only three different recovery states. Such a situation arises quite naturally if one considers, for example, firing, absolutely refractory, and quiescent as the only three recovery states possible. It can easily be shown that if

\[
\begin{pmatrix}
    p_{00} & p_{01} & p_{02} \\
    p_{10} & p_{11} & p_{12} \\
    p_{20} & p_{21} & p_{22}
\end{pmatrix}
\]

for \( 0 < p_{ij} < 1, i,j = 0,1,2, \) there exists a \( k < 1 \) such that \( d<rP,sP> < kd<r,s> \) for all vectors \( r,s \). That is, in one transition the error will be reduced. For a matrix of the type that we have been considering,

\[
\begin{pmatrix}
    p_{00} & 1-p_{00} & 0 \\
    p_{10} & 0 & 1-p_{10} \\
    p_{20} & 0 & 1-p_{20}
\end{pmatrix}
\]

this still holds.
This first series of tests has shown that analyzing the behavior of the lumped model seems to be a reasonable way to study expected base model behavior. We now turn to another series of experiments that has positive feedback between neurons. This series of experiments will continue to investigate a single block.

3.5 Single Block--Positive Feedback

Connection Pattern

The base model has 1 block. The number of neurons ranged from 50 to 1000 and the number of neighbors varied from 20 to 100. [See below] The total input weight [synapse wtx.#nbrs] was held at a constant value of +100.

Parameter Settings

1. Background: -20
2. Noise: Gaussian with mean 0 and standard deviation 10.
3. Threshold: 200e−5

Initial State

All neurons start in state 0.

Input Segment

The external input was 0.

Experiments

<table>
<thead>
<tr>
<th>Size</th>
<th>100</th>
<th>80</th>
<th>60</th>
<th>50</th>
<th>30</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>400</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>800</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
The purpose of this series of experiments is to analyze the effect of feedback within a block of neurons. We wish to show that the lumped model can still explain the behavior of the base model.

The first step is to explain the observed behavior of the lumped model. In our experiments, we used a starting state of all neurons in recovery state 0 (i.e. all firing). The subsequent behavior of the lumped model is shown in fig. 3.5.1. We see that the lumped model firing very quickly reaches a constant value, and from the output, viewed over many time steps, it was observed that the lumped model does reach a steady state. Experimentally, we tried other initial starting states, but in all cases, the identical long term steady state was reached.

![Figure 3.5.1. Lumped Model Output Behavior (positive feedback)](image-url)
One way to analyze the behavior of the lumped model is to view the transition matrices at each time step as representing a non-homogeneous Markov chain. That is, at each time step, the transition matrix is a Markov matrix, but an external input, in this case simply the last percent in state vector, determines the Markov matrix at the next time step. It is easily seen that the following recurrence relations define a non-homogeneous Markov chain.

**Definition 3.5.1**

Let $s(t)$ be the row vector representing the percent in state vector of the lumped model at time step $t$, and let $P(t)$ be the transition matrix at time step $t$. Then, by the recurrence relations we shall mean the following true equations.

1. $s(t) = s(t-1)P(t-1)$
   
   $$s_k(t) = \sum_{i=0}^{6} s_i(t-1)p_{ik}(t-1) \quad k = 0, 1, \ldots, 6$$

2. $x(t) = Ks_0(t)$
   where $K$ is the total feedback and $x(t)$ is the input at time $t$

3. $p_{i0}(t) = F(x(t) + \beta - \theta_i) \overset{\text{def}}{=} F_i(x(t))$
   where $\beta$ is background level and $\theta_i$ is the threshold in recovery state $i$

From equations (1), (2) and (3) we get

4. $p_{i0}(t) = F_i \left[ K \sum_{i=0}^{6} s_j(t-1)p_{j0}(t-1) \right]$}

Recall that
In the following analysis, we shall use the theory of non-homogeneous Markov chains as developed in Paz [31]. We will use the terms "weakly" and "strongly ergodic" which are used by Paz only in this section, and they should not be confused with our previous use of the concept of ergodicity. The following definitions and propositions can be found in Paz [31].

Definition 3.5.2

Let \( P = [p_{ij}] \) be a stochastic matrix. Then define

\[
\delta(P) = \sup_{u,v,j} \Sigma(p_{uj} - p_{vj})^+ \quad \text{where } k^+ \overset{\text{def}}{=} \max\{0,k\}
\]

\[
\|P\| = \sup_{i,j} \Sigma |p_{ij}|
\]

Proposition 3.5.1

\[0 \leq \delta(P) \leq 1\]

Proof:

The proof can be found in Paz [31], page 68.

Definition 3.5.3

A non-homogeneous Markov chain is an infinite sequence of matrices \( \{P_i\}^\infty_{i=0} \) such that the \( P_i \) is the matrix of transition probabilities at time \( t=i \).
Definition 3.5.4

Let \( H_{mn} = \prod_{i=m}^{n} P_i \). Then we say that

(a) A non-homogeneous Markov chain is weakly ergodic if

\[
\lim_{n \to \infty} \delta(H_{mn}) = 0, \quad m = 0,1,2,\ldots
\]

(b) A non-homogeneous Markov chain is strongly ergodic if there exists a constant matrix \( Q \) [i.e. all rows equal] such that

\[
\lim_{n \to \infty} \|H_{mn} - Q\| = 0
\]

Theorem 3.5.1 (Paz)

A Markov chain is weakly ergodic iff \( \exists \) a subdivision of the chain into blocks of matrices \( \{H_{i,j}^{i,j+1}\} \) such that \( \sum_{j=0}^{\infty} (1 - H_{i,j}^{i,j+1}) \) diverges.

Proof:

The proof can be found in Paz [31], page 74.

Proposition 3.5.2

Suppose \( F \) is a strictly monotone increasing c.d.f. (for example, the Gaussian distribution). Then, the lumped model sequence of transition matrices is weakly ergodic.

Proof:

Given the transition matrix, we have

\[
\delta[P(t)] = \max_{0 \leq k \leq 6} [1 - p_{k0}(t)]
\]

From equation (3) we see that

\[
p_{k0}(t) = F(x(t) + \beta - \theta_i) = F(K_0(t) + \beta - \theta_i)
\]

Now since \( F \) is strictly monotone increasing, we have

\[
0 < F(\beta - \theta_i) \leq p_{k0}(t) \leq F(K + \beta - \theta_i) < 1
\]

Thus, \( p_{k0}(t) \) is uniformly bounded away from 0, and if we let the
subdivisions in the previous theorem be just \( H^i_{j,j+1} = P(j) \), then we have \( 1 - \delta(P(t)) \) uniformly bounded away from 0. Unfortunately, weak ergodicity is not very useful. It tells us only that there is a sequence of constant matrices \( \{ E_m \} \) such that

\[
\lim_{n \to \infty} \| H_{mn} - E_{mn} \| = 0
\]

We would like to say that there exists \( \{ E_m \} \) such that

\[
\lim_{n \to \infty} \| H_{mn} - E_m \| = 0
\]

i.e., strongly ergodic. [It can be shown that \( E_m \) is independent of \( m \)].

Strong ergodicity would say that the non-homogeneous process reaches a long run distribution (i.e., steady state). Note, however, that even strong ergodicity would not allow us to say that the same long run distribution was reached independent of the starting state. This is because our non-homogeneous chain is itself a function of the starting state. We have not been able to show that the sequence of transition matrices is strongly ergodic.

We now turn to an analysis of the recurrence relations in an attempt to get some results for the matrix \( P(t) \) as \( t \to \infty \). Although we cannot be sure that there is a limiting matrix, we nevertheless try to get bounds on the entries \( p_{ik}(t) \) as \( t \to \infty \). The following theorem provides us with a method to calculate these bounds (although fairly crude).

**Theorem 3.5.2**

Assume the threshold is a monotone decreasing function. Then,

\[
\lim_{t \to \infty} F_i [(K_P)^t (K_{S_0}(0))] \leq \lim_{t \to \infty} p_{i0}(t) \leq \lim_{t \to \infty} p_{i0}(t) \leq \lim_{t \to \infty} F_i [(K_F)^t (K_{S_0}(0))].
\]

Proof:

Since we assume the threshold function is monotone decreasing, we
have

\[ 0 \leq F_i(x) \leq F_{i+1}(x) \leq 1 \quad i = 0, 1, \ldots, 5 \]

Consequently,

\[ p_{10}(0) = F_i(Ks_0(0)) \]

\[ p_{10}(1) = F_i \left[ \sum_{i=0}^{6} s_i(0) F_j(Ks_0(0)) \right] \]

\[ \geq F_i \left[ KF_0(Ks_0(0)) p_{j0}(1) \right] \]

\[ p_{10}(2) = F_i \left[ \sum_{i=0}^{6} s_i(1) p_{j0}(1) \right] \]

\[ \geq F_i \left[ K \sum_{i=0}^{6} s_i(1) F_j[KF_0(Ks_0(0))] \right] \]

\[ \geq F_i[KF_0(Ks_0(0))] \]

Inductively,

\[ p_{10}(t) \geq F_i[(KF_0)^t(Ks_0(0))] \]

Similarly, we can get an upper bound

\[ p_{10}(t) \leq F_i[KF_0]^t(Ks_0(0))] \]

Since \( F_i \) is monotone, we have either

\[ KF_i(x) \geq x \]

\[ \therefore (KF_i)^t(x) \geq (KF_i)^{t-1}(x) \geq (KF_i)^{t-2}(x) \ldots \geq CF_i(x) \geq x \]

or

\[ (KF_i)^t(x) \leq (KF_i)^{t-1}(x) \ldots \leq CF_i(x) \leq x \]

In either case since we have a bounded monotone sequence, the limit exists as \( t \to \infty \) [may depend on \( KS_0(0) \)]. Thus, the result follows.

We now present an example of calculating such bounds. (Note that the limit points are fixed points of \( KF_0 \) or \( KF_i \)). Let \( K = 100, \sigma = 20, \)

\[ \theta_i = 200e^{-i}, \mu = 0, \beta = -20. \]

\[ \lim_{t \to \infty} (KF_0)^t(Ks_0(0)) = 0 \]

since this is the only fixed point of \( KF_0 \). Also, \( \lim_{t \to \infty} (KF_6)^t(Ks_0(0)) = 100. \)
Thus, \( F_1(0) = P[-20 - 200/e] = 0 \) and \( F_1[100] = .74 \). So, \( 0 \preceq p_{10} \preceq .74 \) is the limit. Other bounds for this situation yield \( 0 \preceq p_{00} \preceq 0 \) and \( .02 \preceq p_{60} \preceq 1 \).

It is clear that such bounds do not provide too much information. We have seen that analytic results for long run distributions appear to be very difficult to obtain in the general case. Rather than continuing to attack this problem we now turn to more informal "proofs."

We now consider a heuristic "proof" that the lumped model with the parameters of this series can in fact reach one and only one steady state independent of the initial state, if it reaches a steady state at all.

We assume that in the steady state, the first six recovery states \( 0, \ldots, 5 \) have approximately the same percent in state values. Moreover, nearly all the neurons that do fire will in fact have been in recovery state 6. Therefore, if \( s \) represents the steady state vector,

\[
s = <s_0, s_0, s_0, s_0, s_6> \quad \text{and} \quad s_6 = 1 - 6s_0
\]

Assuming that \( p_{60}s_6 = s_0 \), we get

\[
p_{60} = s_0/s_6 = s_0/(1 - 6s_0)
\]

Now,

\[
p_{60} = F_6[Ks_0] = \left( \frac{Ks_0 - \theta_6 - \beta}{\sigma} \right) = \left( \frac{100s_0 - 20.5}{10} \right)
\]

So

\[
\phi \left( \frac{100s_0 - 20.5}{10} \right) = \left( \frac{s_0}{1-6s_0} \right) \quad \text{[Note: \( \phi \) is the standard normal c.d.f.]} \]

is satisfied in the steady state. Graphing each side of the above equation as a function of \( s_0 \) (see fig. 3.5.2), we find that the only possible solution \( s_0 \) is very close to the observed value.

The informal approach has thus given us useful information.
Figure 3.5.2. Heuristic Calculation of Lumped Steady State. (Intersection of the two curves is the steady state firing level.)
Since we wish to use the deterministic lumped model to predict the behavior of the probabilistic base model, we now informally analyze how error might propagate. Recalling that what might happen is that the true base model percent in state vector is a slightly displaced version of the predicted percent in state vector, we examine what happens to two lumped model states that are close to each other. Instead of using the seven-dimensional vector that we actually have, we make the problem more tractable by just using two recovery states 0 and 6. Thus, we consider the next state transition of the lumped model, starting with various values in states 0 and 6. If the entries do not add up to 1, we uniformly spread the remaining percentage in states 1 through 5.

This gives us the "force field" in fig. 3.5.3. Surprisingly, an analysis of this force field is a mine of information. Although the force field is not quite accurate since we do not have a homomorphism from the seven-dimensional state space to the two-dimensional one, nevertheless, the force field gives us a good understanding of the behavior of the lumped model. The steady state point is clearly locatable. Also, we observe that there are some states which diverge quite significantly. Consequently, if a base model received enough noise to push it far enough away from the steady state, then the trajectory of the base model should follow the force field trajectory defined by the lumped model.

To see whether this was actually true, we graphed several base model runs. We first simply plotted the firing level of runs with 400 neurons and 50 neighbors [fig. 3.5.4]. On the surface, this does not appear to match in any way the lumped model run [fig. 3.5.1] which is also a plot of the firing level vs. time. However, we decided to plot the same base runs using the same two-dimensional "state space" as the
Figure 3.5.3. Lumped Model Two Dimensional Force Field (positive feedback)
"force field" of the lumped model. This plot [fig. 3.5.5] reveals quite clearly that once the base model is displaced from equilibrium (by noise) it tends to follow the "force field" of the lumped model. We plotted other base model runs using the same two-dimensional "state space." [Figs. 3.5.6-3.5.12] They show the qualitative similarity between base model runs and lumped model runs. We see that when the neighborhood size is reasonably large [figs. 3.5.6, 3.5.10, 3.5.11] the base model spends much of its time near equilibrium. When it is displaced by noise it follows a lumped model trajectory that is not too large. Also, it seems that there is almost a critical neighborhood size in the sense that even with 1000 neurons, if the neighborhood size is small, for example, 20, 30 and 50 neighbors [figs. 3.5.7-3.5.9] then there is always sufficient noise to kick the base model away from equilibrium.

The concept of a "force field" has provided us with a convenient way to use the deterministic lumped model. It is seen that the lumped model can be used for general predictions of long term behavior. The force field also provides a qualitative conditional prediction of future base model behavior in the short term. We also see that for more accurate behavior matching we would have to consider a probabilistic lumped model that is, in an appropriate sense, a homomorphic image of the base model. We shall be able to do this in the next chapter when we apply the concepts of ergodic machines to homomorphic simplifications and consider the concept of a p-homomorphism. In the next section, however, we continue the analysis with a deterministic lumped model, relying heavily on the idea of investigating the "force field" behavior of the lumped model.
Figure 3.5.5a  Base Model Two Dimensional Runs  400 Neurons  
50 Neighbors (Each point represents a time step, 
two different runs shown)
Figure 3.5.5b  Base Model Runs  400 Neurons  50 Neighbors (2 runs)
Figure 3.5.6. Base Model Runs 400 Neurons 100 Neighbors (2 runs)
Figure 3.5.7. Base Model Run 1000 Neurons 20 Neighbors
Figure 3.5.8. Base Model Run with 1000 Neurons 30 Neighbors
Figure 3.5.9. Base Model Run 1000 Neurons 50 Neighbors
Figure 3.5.10  Base Model Run  400 Neurons  60 Neighbors
Figure 3.5.11. Base Model Run - 800 Neurons 60 Neighbors
Figure 3.5.12. Base Model Runs 50 Neurons 50 Neighbors (2 runs)
3.6 Two Blocks--A Flip-Flop

Connection Pattern

The base model had two blocks. The number of neurons ranged from 50 to 500 per block and the number of neighbors ranged from 20 to 50. [See below for exact values for various runs.] The total input weight to each block from the other block was held at a constant value of -100. There was no feedback from a block to itself.

Parameter Settings (for each block)

1. Background: 50
2. Noise: Gaussian with mean 0 and standard deviation 20.
3. Threshold: 27e^-5

Initial State

The initial state varied depending on the purpose of the run.

Input Segment

The input segment varied depending on the run.

Experiments

Experiment 1 (setting the Flip-Flop)

<table>
<thead>
<tr>
<th>NBRS Size</th>
<th>50</th>
<th>40</th>
<th>20</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Experiment 2 (stability)

<table>
<thead>
<tr>
<th>NBRS Size</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>100</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>500</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
Experiment 3 (reset)

<table>
<thead>
<tr>
<th>Size</th>
<th>NBRS</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The purpose of this series of experiments was to examine the base-lumped modelling pair in a more complex setting than has been previously considered. We have seen that analytic results are extremely difficult to obtain even in the case of a single block. Thus, when we now consider two interconnected blocks, we shall rely on experimental studies of the lumped model behavior to give us insight into predicting behavior of the base model. The two blocks were interconnected, and the parameters chosen, so that a "flip-flop" behavior was obtained. Each block negatively inhibited the other block with a total input weight of -100, and had no effect on itself. Thus, for example, when Block 1 was firing at a high level and Block 2 was firing at a low level, this situation would theoretically persist until there was some external input. This firing condition constitutes the set state of the flip-flop. If, subsequently, a high level of external input was supplied to the second block, then the reverse situation should be obtained, with Block 2 firing at a high level and Block 1 firing at a low level. This state of the flip-flop is called the reset state. Clearly, the flip-flop is able to remember the last high level of external input that it received. Given the structure of the flip-flop there is one additional state that occurs. This is when both blocks alternately fire at a high level and then at a low level, in synchrony. We shall term this flip-flop state as lockstep.
We found that the lumped model indeed exhibited the three stable states of set, reset, and lockstep. Reset, of course, is simply a complementary state to the set state. Lockstep, in the lumped model, seemed to be an extremely stable state, since it took a fairly high level of input to get out of this state. A qualitative examination of base model behavior revealed no surprising effects. The base model exhibited a stable lockstep state and also could be put into the states set and reset. For the base model, however, the set and reset states were only conditionally stable. That is, after several time steps, depending on the number of neurons and neighbors, and varying with a particular probabilistic run, the base model would eventually go into the lockstep state. We shall see the reasons for this behavior, as well as other lumped model behavior, as we examine the base model using detailed considerations of lumped model behavior.

The first consideration of interest is the level of continuous external input necessary to set the flip-flop. That is, starting from the lockstep state, which was Block 1 and Block 2 synchronously alternating with the percentage vectors \(<97.77, 0, 2.2, 0, .01, 0, 0>\) and \(<.01, 97.77, 0, 2.19, .01, 0, 0>\), what level of external input is necessary to set the flip-flop? We found that the minimum continuous external input necessary to Block 1 was 47. Now, the base model lockstep state is predicted to be quite stable (more on this later on), and so we would expect that essentially the same amount of external input would be necessary to set the base model. At most, with a small number of neurons and neighbors, it might be slightly easier to set because of the random noise generated by such a configuration. A detailed analysis of this situation was not deemed necessary. Experiment 1 shows the actual runs
considered. For the case of 50 or 40 neighbors, we found that an input level of 50 was necessary to set the base model, and for the case of 20 and 10 neighbors, we found that an input level of 45 was sufficient. Since we considered increments of 5 for the external input, the results are exactly what one might predict.

A more interesting investigation using the lumped model was termed exploring the regions of stability. As has been previously mentioned, the lumped model exhibits the flip-flop states of set and lockstep. The set state occurs with Block 1 having the percentage vector \(<85.38, 14.13, .48, 0, 0, 0, 0>\) and Block 2 having the percentage vector \(<3.53, 3.53, 3.49, 3.40, 3.28, 3.16, 79.58>\), in the steady state. Now, it seems that in the set state, most of the information is captured by considering the percentages in neuron states 0 and 1 for Block 1 and the percentages in neuron states 0 and 6 for Block 2. Effectively, we are considering an approximate lumping of the original state sets to one where there are only two neuron states in Block 1 and Block 2. Now, let us call this lumped version of the original lumped model the approximate lumped model. Suppose we now put the approximate lumped model in various initial states, that is, have the original lumped model starting in various percentages firing (i.e., in neuron state 0) and the rest of the percentage firing to be in neuron state 1 for Block 1 and neuron state 6 for Block 2. Without any external input, does the approximate lumped model go to the flip-flop state of set or reset or lockstep? Fig. 3.6.1 is a partial summary of the results. Fig. 3.6.2 shows the entire region considered, calculated on the basis of symmetry. The advantages of essentially only looking at neuron states 0 and 1 for Block 1 and 0 and 6 for Block 2 is that the behavior is now much easier to visualize and yet
Figure 3.6.1. Partial Region of Stability Graph
L- Lockstep is final result
S- Set or Reset is final result
Figure 3.6.2. Entire Region of Stability Graph
captures the essential features. It is apparent that the region of
stability, that is, those initial states leading to a set state (or re-
set state), is quite small compared to the lockstep region.

Since the lumped model, starting in an initial state as indi-
cated, essentially goes in one time step to a state that can be similarly
aggregated, that is, most of the percentages are in neuron states 0 and 1
for Block 1 and neuron states 0 and 6 for Block 2, we can consider that
an "approximate homomorphism" exists between the lumped model and approx-
imate lumped model, and consequently we can get a "force field" as in
fig. 3.6.3. This "force field" gives an approximate representation of
how the lumped model behaves as it goes from an initial state to lockstep
or an initial state to set. Although the "force field" is only approxi-
mate, nevertheless, it shows why the lockstep state is so stable. We see
that once the lumped model is in lockstep, it has extreme changes of
neuron states. Consequently, there is a very strong vector from one of
the lockstep phases to the other. Recall that in lockstep, both blocks
synchronously alternate with the approximate percentage vectors of
\(<97.77, 0>\) and \(<.01, 97.77>\) for Block 1 and \(<97.77, 0>\) and \(<.01, 0>\) for
Block 2. Of course, our stability diagram of fig. 3.6.1 assumes even
less information, for the percentages not firing are assumed to be in
neuron states 1 and 6 for Blocks 1 and 2 respectively. In any case,
since we know that the base model would be expected to follow the lumped
model trajectories it is apparent that the chances of the base model
deviating from lockstep sufficiently to become set are extremely remote.
It would take a tremendous level of noise in just the right amounts over
several time steps to get the base model into the set or reset region.
This is the justification for considering lockstep to be a very stable
Figure 3.6.3. Lumped Model Two Dimensional Force Field for Flip-Flop
(arrow represents movement at next time step)
base model flip-flop state, and, in fact, the base model was never observed to have become set autonomously, once it was in lockstep.

On the other hand, it is clear that if the base model is in a set state, the probabilistic nature of the base model would cause the trajectories to "wander around" the set point, or the point of equilibrium, using the effective force field as shown previously in fig. 3.6.3. From figs. 3.6.2 and 3.6.3, it would seem that it would be fairly easy for the noise of the probabilistic base model to be thrown into a region of lockstep after which it would inexorably go to the very stable lockstep state. Thus, we would predict that the base model could only be conditionally stable in the set flip-flop state and that after several time steps, the base model would slip into the lockstep state. This is, in fact, what does occur. Table 3.6.1 is a summary of the information gathered from the runs indicated in Experiment 2. The base model was run in cycles of 24 so this is the longest time period of stability that was considered. As can be seen, the effect of a small number of neuron neighbors is quite pronounced since even with 500 neurons in the base model, with only 20 neighbors per neuron, the base model is rarely stable for the full 24 time steps possible. On the other hand, with 50 neighbors per neuron, even 100 neurons is fairly stable. The case of 500 neurons and 50 neighbors indicates that once there is a reasonable neighborhood size, with enough neurons, the base model very quickly begins to be extremely stable.

To get a further idea as to how the base model decays, we decided to find the minimum input level to the lumped model that would unset it, once the lumped model was set. We decided to try input level of various lengths rather than a continuous input. Thus, we used an input of lengths
TABLE 3.6.1
BASE MODEL UNSET TIMES FROM THE SET STATE

<table>
<thead>
<tr>
<th>Block Size</th>
<th>Number of Neurons</th>
<th>Number of Neighbors</th>
<th>Number of Runs that Unset After x-y Time Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>50</td>
<td>20</td>
<td>1-4 5-8 9-12 13-16 17-20 21-24 24+</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>50</td>
<td>3 4 2 0 0 1 3</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>20</td>
<td>1 1 1 0 1 0 6</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>50</td>
<td>3 9 2 1 1 1 1</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>20</td>
<td>2 1 1 1 0 1 4</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>50</td>
<td>0 0 0 0 1 1 8</td>
</tr>
</tbody>
</table>

NOTE: Base Model was run in periods of 24 so this is the longest possible time to stay set.

1, 2, 3, 4, and 5. After the appropriate input was applied for the proper number of time steps (1-5), it was removed, and the lumped model was observed to see whether it went into lockstep or back to the set state. Table 3.6.2 shows the minimum input of the various lengths needed to unset the lumped model. Thus, although an external input of 16 for one time step (length one) unsets the lumped model, if we apply the external input for three time steps, we need only apply an external input of magnitude 11. It may seem surprising that if the input is applied for two time steps, then an input level of 18 is necessary, but this is reasonable when we reexamine the force field of fig. 3.6.3. Since the first input displaces the lumped model away from equilibrium, the lumped model, following the force field, would react by moving past the equilibrium point in the other direction. A hit at this time step, then, would only tend to knock the lumped model closer to equilibrium rather than farther away.
TABLE 3.6.2
MINIMUM INPUT NECESSARY TO LUMPED MODEL TO UNSET IT

<table>
<thead>
<tr>
<th>Input Length</th>
<th>Minimum Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
</tr>
</tbody>
</table>

Given the data gathered, it might be desirable to attempt to analytically predict when the base model would unset. Unfortunately, because of the many complex trajectories that the base model might follow during unsetting, an analytical analysis is not feasible. One might expect that it would be possible to predict how frequently a single input due to noise might unset the base model, but even this is intractable analytically. We shall see in the next chapter that the way to proceed is to study a probabilistic lumped model which can properly predict the base model behavior. Experiments with such a probabilistic lumped model will yield the type of information that is extremely difficult to obtain analytically.

The final set of experiments associated with the flip-flop model, Experiment 3, investigated the "reset" behavior of the base-lumped model pair. Once the lumped model was set, it was found that a continuous input of 35 was sufficient to reset it. Thus, although a level of 47 is necessary to set the lumped model from lockstep, a lower level of external input suffices for putting it into the reset state once it is set. This is plausible, because the state trajectory followed during resetting
need never actually go through the true lockstep states of both blocks synchronously having the percentage vectors <97.77, 0, 2.2, 0, 0.01, 0, 0> and <0.01, 97.77, 0, 2.2, 19, 0, 0.01, 0>.

Upon observing the region of stability of fig. 3.6.2 and the force field of fig. 3.6.3, it appears that the specific trajectory that the lumped model would follow in resetting would be quite critical, and that small deviations might in fact lead to the lockstep state. Thus, when trying to predict base model behavior, it would seem that there would be a well-defined fraction for which the base model would go into lockstep instead of getting set depending on the input level. Table 3.6.3 summarizes the information obtained for reset tests with input levels of 25 and 30. We see that resetting occurs consistently at an input level of 30 which is a little less than the level of 35 predicted by the lumped model. This again would be expected since it was found to be somewhat easier to set the base model from lockstep also. The input level of 25 shows that the effect of a small number of neighbors is again to make it easier for the flip-flop to reset, just as it was easier to initially set the flip-flop from lockstep.

<table>
<thead>
<tr>
<th>Block Size</th>
<th>Number of Neurons</th>
<th>Number of Neighbors</th>
<th>Input = 25</th>
<th>Input = 30</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reset</td>
<td>Lockstep</td>
</tr>
<tr>
<td>50</td>
<td>20</td>
<td></td>
<td>37</td>
<td>23</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td></td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>100</td>
<td>20</td>
<td></td>
<td>47</td>
<td>18</td>
</tr>
<tr>
<td>100</td>
<td>50</td>
<td></td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>500</td>
<td>20</td>
<td></td>
<td>31</td>
<td>0</td>
</tr>
</tbody>
</table>
The analysis of the flip-flop case has shown that a detailed analysis of the lumped model can reveal much important information about base model behavior. Furthermore, the concepts of looking at the force field of the lumped model seems to have general usefulness, as we also had used this technique in analyzing the single block case. However, it appears that more fruitful analysis can only be obtained by a more representative lumped model than simply the deterministic lumped model that we have been considering.

3.7 Complexity

It is apparent that the lumped model that we have been considering as the approximate homomorphism of a base model is much less complex than the base model. The simplicity of the lumped model relative to the base model was apparent in the computer implementation. The simulation programs for the lumped model were far simpler and shorter than those for the base model. The lumped model simulation used less computer memory and also required less computer time per simulated time step. A comparison between running times for the lumped and base model can be found in table 3.7.1 and figures 3.7.1 and 3.7.2.

3.8 Conclusion

The experimental investigations show the usefulness of the concept of approximate homomorphisms, although they demonstrate the need for longer runs in the case of positive feedback to distinguish between cyclic and aperiodic behavior. In the next chapter, we return to the formalism and show the derivation of p-homomorphic lumped models, and give some results with probabilistic lumped models for comparison. In chapter V, we look at the experimental results again in the context of chapter IV.
### TABLE 3.7.1

**COMPLEXITY ANALYSIS OF BASE MODEL VS. LUMPED MODEL**

(Running times for base model in second series of experiments)

<table>
<thead>
<tr>
<th>Neighbors</th>
<th>25</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>400</th>
<th>800</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.7</td>
<td>4.2</td>
<td>5.1</td>
<td>6.4</td>
<td>9.6</td>
<td>16.15</td>
</tr>
<tr>
<td>10</td>
<td>3.75</td>
<td>4.25</td>
<td>5.55</td>
<td>8.2</td>
<td>13.4</td>
<td>23.75</td>
</tr>
<tr>
<td>20</td>
<td>3.85</td>
<td>5.05</td>
<td>6.75</td>
<td>10.4</td>
<td>17.8</td>
<td>32.55</td>
</tr>
<tr>
<td>40</td>
<td>--</td>
<td>5.55</td>
<td>8.8</td>
<td>14.9</td>
<td>26.5</td>
<td>49.55</td>
</tr>
<tr>
<td>80</td>
<td>--</td>
<td>--</td>
<td>12.95</td>
<td>23.6</td>
<td>42.95</td>
<td>--</td>
</tr>
</tbody>
</table>

**NOTES:** Values given are average length of time required per simulated time step (in seconds).

For comparison, the lumped model requires only 2.75 seconds per timestep--regardless of the number of neurons or neighbors.

We would expect the running time of the base model to be of the form $T = K+N(an+b)$ since the main loop of the program is:

- Loop over all neurons (N)
  - Computer threshold and external input
    - Loop over neighbors of this neuron (n)
      - Sum up inputs
      - Compare input and threshold

In fact, $K = 3.2$, $a = .001$, $b = .015$ gives a rather good fit. See figs. 3.7.1 and 3.7.2.
Figure 3.7.1. Running Time Complexity
CHAPTER IV

PROBABILISTIC HOMOMORPHISMS AND SIMPLIFICATIONS

In the previous chapter, we have shown how a probabilistic base model could be modelled by a deterministic lumped model. We claimed that the lumped model that was considered was an "homomorphic" image of an ideal base model. The ideal base model had an infinite number of components and an infinite number of neighbors per component. It was also suggested that for a finite number of components in the base model, an appropriate probabilistic lumped model could correctly simulate base model behavior. We now attempt to formalize these concepts, using the idea of an ergodic machine that has already been developed. Recall that an ergodic machine is a way to transform a probabilistic model into a deterministic one. Since we know what we mean by simplification in the deterministic context, we shall now be able to use the same idea in the probabilistic context. We shall also see that treating the base model as an ergodic machine will give us insight into how to simplify. The first point that we investigate is the concept of a probabilistic homomorphism.

4.1 Probabilistic Homomorphisms

The intuitive idea of a probabilistic homomorphism is very simple. Suppose we have a class of deterministic models corresponding to any probabilistic model A. Call this class $C(A)$. Then, if there is a homomorphism (the usual deterministic idea) from an element of $C(A)$ to
an element of $C(B)$ for some probabilistic model $B$, then $B$ will be considered to be a probabilistic homomorphism of $A$.

The first thing that we must do is to delineate the class of models that we are talking about. Therefore, since the probabilistic models we are using are probabilistic automata, we shall let $A$ be a probabilistic automaton. Now, we have shown in Chapter II that under very general conditions, there is a class of ergodic machines that faithfully represents $A$. In fact, if $A$ has a finite state space, all that is necessary is that for each external input $x$, if $\mu(x)$ represents the minimum non-zero transition of the probabilistic transition matrix $[p_{ij}(x)]$ then $\min_x \mu(x) > 0$. In Chapter II, we also investigated cross-products of ergodic machines and permutations of the seeds of the cross-product machine. We wish to expand the representative class of an automaton $A$ by including machines with permutations. We are thus led to the following definition.

Definition 4.1.1

Let $E$ be an ergodic machine that is the cross-product of ergodic machines $E_i$, that is, $E = \bigotimes_{i \in \mathbb{N}} E_i$, where $\mathbb{N}$ is an at most countable set. Let $\{\pi(t)\}$ be a time indexed set of permutations, with $\pi(t)$ a permutation (bijective map) on $N$. Then a generalized ergodic machine is $E$ together with $\{\pi(t)\}$ and the deterministic machine that is represented is as before except that at each time step, $\pi(t)$ is applied to the vector of seeds $\bigotimes_{i \in \mathbb{N}} T_i$.

Notice that we are essentially allowing any arbitrary permutation at each time step. The permutation may in fact be determined by the state of $E$ at time step $t$. For convenience we repeat definition 2.10.4 at this point.
Definition 4.1.2 (also 2.10.4)

A faithful transformation $\zeta = \chi_{i=1}^N \zeta$ is permutation independent if the information loss function is strong for every vector sequence allowing an arbitrary permutation of the seed vector $<r_1, \ldots, r_N>$ at each time step. We say that $\zeta$ is permutation independent if for every $N$, $\zeta$ is permutation independent.

Since the information loss function is strong even with permutations, it is clear that a faithful ergodic machine with permutations should also be faithful (see Definition 2.6.5). We thus have the following.

Definition 4.1.3

Let an ergodic machine $E$ with transformation $\zeta = \chi_{i \in N} \zeta$ for some $N$ faithfully represent an automaton $A$. Then, if $\zeta$ is permutation independent, the generalized ergodic machine $<E, \pi(t)>$ also faithfully represents $A$.

Definition 4.1.4

The representative class $C(A)$ of a probabilistic automaton $A$ is the class of generalized ergodic machines that faithfully represent $A$.

We are now in a position to define what we mean by a probabilistic homomorphism.

Definition 4.1.5

A probabilistic automaton $B$ is a probabilistic homomorphic image of a probabilistic automaton $A$ if there is a homomorphism $h$ (the classical deterministic homomorphism) from an element of $C(A)$ to an element of $C(B)$. We shall also say that $h$ is a probabilistic homomorphism.
(p-homomorphism) from A to B.

Our definition has captured the intuitive notion. We can effectively regard any faithful representative of the automaton A as just as good as any other. Thus, if we can simplify one representative A' homomorphically to get B' and then choose a more convenient representative B, we would want to say that B is a simplification of A.

We now consider simplifying base models using these ideas. Given a base model-lumped model pairing, what we need to do is to represent each of them as (possibly generalized) ergodic machines and find a (deterministic) homomorphism between the two deterministic representatives. We shall develop the simplification results in a general context but it will be useful to keep the neural network model in mind. We shall use that model as a continuing example to clarify the exposition. Thus, the simplifications that we shall consider will involve the percentage of components in a particular component state. For example, in the neural network model, we only aim to preserve the percent in state vector representing the recovery states.

We begin by considering the infinite case base model and formalizing it as an ergodic machine. We do the same with the lumped model and then exhibit a map \( h \), which is a homomorphism from the state space of the base model to the state space of the lumped model. The homomorphism of course allows us to say that the base model and lumped model are behaviorally equivalent. We shall at first assume that each component of the base model receives the same input as every other component. Such an hypothesis is justified in the neural network case if we have an infinite number of neighbors by an application of the law of large numbers. In the case of a finite number of components per aggregation, this
hypothesis would be correct under some connection patterns and would be a good approximation if the number of neighbors is sufficiently large. We shall investigate in a later section what happens when this hypothesis is removed.

4.2 Simplification of the Ideal Base Model

We model an individual component as an ergodic machine \( M_i = \langle X_i, X_i, Y_i, Z_i, \delta_{li}, \lambda_{li} \rangle \) where each component is identical. Of course, the initial seeds of the components will be chosen independently from the distribution represented by \( \Pi_i \). We can represent an infinite base model pool as a countable cross-product of the individual ergodic machines. (We only deal with countably infinite numbers of components.) Because of our assumption about inputs, we do not have to specify the connection pattern. Instead, we assume that the input to each neuron is some function \( f \) of the state of the cross-product machine at the previous time step.

Let the infinite pool be represented by \( M = \bigoplus_{i=1}^{\infty} M_i \). A typical state of \( M \) is thus \( (s_1, r_1) \times (s_2, r_2) \times \ldots \) and the transition function is defined by componentwise operation on \( M_i \). Alternatively, we may represent the state space by the cross-product of two infinite vectors \( <s_1, s_2, s_3, \ldots> \times <r_1, r_2, r_3, \ldots> \). The elements of \( r_1 \) are of course the seeds. The initial value of the seeds is unknown to us, and different configurations of initial seeds will give rise to different pseudo state trajectories. The output function for the ideal base model is what we are interested in preserving under simplification. We have already decided that we wish to preserve the percentage of components in each component state. Consequently, to define the output function \( \lambda \), we must
decide what we mean by proportions of components in a component pseudo-state \( s \). We proceed as follows. Given the pseudo-state vector \(<s_1, s_2, s_3, \ldots>\) we define the indicator functions \( I_k \) for each component pseudo-state \( k \) by

\[
I_k(s_i) = 1 \quad \text{if } s_i = k \\
= 0 \quad \text{otherwise for } k = 1, \ldots, n
\]

where \( n \) is the number of pseudo-states of a component. \( I_k \) indicates whether or not its argument is in pseudo-state \( k \) or not. Now, let

**Definition 4.2.1**

\[
q_k = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} I_k(s_i) \quad k = 1, \ldots, n
\]

We claim that the idea of proportions of components in a particular pseudo-state \( k \) to make sense is equivalent to the existence of this limit. We shall assume that this limit exists initially for the base model ergodic machine for every \( k \). We will show that the limit then still exists at the next time step. The definition of \( \lambda \) is now apparent. Given the state \(<s_1, s_2, \ldots> \times <r_1, r_2, \ldots>\), we have

\[
\lambda[<s_1, s_2, \ldots> \times <r_1, r_2, \ldots>] = <q_1, \ldots, q_n>
\]

where \( q_k \) is defined above. This completes the definition of the ideal base model. In the neural network case we have:

**Example 4.2.1**

We represent a neuron as an ergodic machine. Suppose we have the threshold function \( 27e^{-S} \), and a noise that is \( N(0,10) \), that is, Gaussian with mean 0 and standard deviation 10. Then we can represent the neuron as \( \text{NEURON} = <X, S, \Pi, Y, \zeta, \delta_1, \lambda_1> \) where

(a) \( X \) = some set of real numbers. We assume that the maximum and
minimum values of the input are bounded, that is, that the input is normalized. Thus $X$ has a finite maximal element and minimal element.

(b) $S = \{0, 1, 2, 3, 4, 5, 6\}$ 0 represents the neuron firing or emitting an output pulse, 1 represents not firing for 1 time step, etc.

(c) $\mathbb{P} = \langle R, \mathcal{B}, P \rangle$ where $R$ is the unit interval $[0, 1)$, $\mathcal{B}$ is the set of Borel sets on $[0, 1)$ and $P = dx$, the Lebesgue measure on $R$.

(d) $Y = S$ as the output space.

(e) $\zeta$ is a weakly mixing transformation on $R$ that is faithful.

(f) Let $F$ be the cumulative distribution function for a $N(0, 10)$ random variable. Then,

$$\delta_1(x, s, r) = \begin{cases} 
0 & \text{for } F^{-1}(r) < x - 27e^{-s} \\
= s + 1 & \text{otherwise unless } s = 6 
\end{cases}$$

whence $\delta_1(x, s, r) = s$

(g) $\lambda_1(s, r) = s$

Notice that since we assume a maximum possible input (and a minimum), the seed space is properly divided up so that there is a minimum measure over all possible transition seed spaces over all input values. Thus, we can find a faithful weakly mixing transformation of the multiplicative type by Theorem 2.7.1. Corresponding to an input value $x$, there is a well defined set $A_{s_1, s_2}(x)$ which is the transition seed space from pseudo-state $s_1$ to pseudo-state $s_2$. $A_{s, 0}(x)$ will be especially important. The base model neuron block can now be represented by an infinite cross-product of the NEURON defined above. Several blocks can be similarly represented, but for convenience, we consider the single block case only. This completes the example.
We now turn to an examination of the formal definition of the lumped model of a single block.

The lumped model will be represented by the ergodic machine

\[ M' = \langle X', S', \Pi', Y', \zeta', \delta'_1, \lambda' \rangle. \]

(a) \( X' \) represents the input set. We shall assume that since the input was identical to each component it can in fact be determined from the percent of components in a particular component pseudo-state. That is, the lumped input is precisely the identical component input of the base model.

(b) \( S' = I^n \) where \( I \) is the real interval \([0,1]\) and \( n \) is simply the number of base model component pseudo-states, assumed to be finite. A typical lumped model pseudo-state is thus \( \langle p_1, p_2, \ldots, p_n \rangle \) where each \( p_i \) represents the percent of base model components in base component pseudo-state \( i \).

(c) \( \Pi' \) is a probability space on a trivial one-point space, say \( \{0\} \). Thus, \( P[\{0\}] = 1 \).

(d) \( Y' = S' \) and \( \lambda'_1[\langle p_1, \ldots, p_n \rangle \times 0] = \langle p_1, \ldots, p_n \rangle \).

(e) \( \zeta' \) is the trivial mapping \( \zeta'(0) = 0 \). Notice that \( \zeta' \) is still an ergodic transformation, although trivially so.

(f) We need only define the transition function. In the base model, for each component, given an input \( x \), for all pairs of states \( i, j \), the transition seed space (Def. 2.5.1) \( A_{i,j}(x) \) is a well-defined set. Since the space \( R \) of the base model component is a probability space, we let \( p_{ij}(x) = \text{Prob}[A_{i,j}(x)] \). Let \( P(x) \) be the matrix \( [p_{ij}(x)] \). Then, we define \( \delta'_1[x, \langle p_1, \ldots, p_n \rangle, 0] = \langle p_1, \ldots, p_n \rangle \cdot P(x) \).

Notice that we have really defined a deterministic machine with-
out any hidden components since the pseudo state is equivalent to the actual state. Since we do not use any property of the lumped model related to ergodic machines, we shall not be concerned that the pseudo state space is not countable.

Before we define a homomorphism from the base model to the lumped model we consider the lumped neural network model.

**Example 4.2.2**

We continue the example of 4.2.1 to the lumped case. Thus, $\Pi'$, $\xi'$, $\lambda'_1, Y', X'$ are as in the general case. $S' = I^T$ and a typical state is $<p_0, \ldots, p_6>$. Finally, the transition function is defined using the familiar matrix of Chapter III to be

$$P(x) = \begin{bmatrix}
p_{00}(x) & 1-p_{00}(x) & 0 & \cdots & 0 \\
p_{10}(x) & 0 & 1-p_{10}(x) & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
p_{60}(x) & 0 & 0 & \cdots & 1-p_{60}(x)
\end{bmatrix}$$

In this case $p_{10}(x)$ is the probability of the transition seed space $A_{10}(x)$. We notice that this is the set of seeds $r$ such that $F^{-1}(r) \leq x-27e^{-S}$ where $F$ is as in Example 4.2.1. The probability of this set is clearly $F(x-27e^{-S})$. (See the matrix in section 3.4.) This concludes the example.

We shall now consider the homomorphism from the ideal base model to the lumped model, both of which are represented by deterministic machines.
Definition 4.2.2

Let \( \chi(s_i, r_i) \) be a state of the base model. Define the mapping \( h \) to be

\[
h\left[ \chi(s_i, r_i) \right] = [q_1, \ldots, q_n, 0]
\]

where \( q_i \) is as in Definition 4.2.1.

The proof that \( h \) is in fact a homomorphism is quite simple. First, however, we make precise our hypothesis on the initial values of the seeds \( <r_1, r_2, \ldots> \). Recall that we initially chose them independently and identically distributed. Equivalently, we can say that for any subsequence \( r_{i_1}, r_{i_2}, r_{i_3}, \ldots \)

\[
\lim_{N \to \infty} \frac{1}{N} \sum_{k=1}^{N} I_A(r_{i_k}) = P(A) \quad \text{a.e.}
\]

\( I_A \) is of course the indicator function of the set \( A \). We assume that the above condition exists initially. We wish to show that this condition is preserved over time.

Lemma 4.2.1

Suppose that the above hypothesis holds at time \( t \). Then, if \( <r_1, r_2, \ldots> \) is the seed vector at time \( t \), the above holds also for \( <\zeta(r_1), \zeta(r_2), \ldots> \), that is, for the seed vector at time \( t+1 \).

Proof:

We wish to show that for any subsequence \( \zeta(r_{i_1}), \zeta(r_{i_2}), \ldots \)

\[
\lim_{N \to \infty} \frac{1}{N} \sum_{k=1}^{N} I_A\left[ \zeta(r_{i_k}) \right] = P(A) \quad \text{a.e.}
\]

Now,

\[
\lim_{N \to \infty} \frac{1}{N} \sum_{k=1}^{N} I_A\left[ \zeta(r_{i_k}) \right] = \lim_{N \to \infty} \frac{1}{N} \sum_{k=1}^{N} I_{\zeta^{-1}(A)}(r_{i_k})
\]

\[
= P[\zeta^{-1}(A)] \quad \text{(by hypothesis)}
\]

\[
= P[A] \quad (\zeta \text{ is measure preserving})
\]
Notice that we only needed the measure preserving property of the transformation for the proof.

Theorem 4.2.1

The mapping $h$ of Definition 4.2.2 is a homomorphism.

Proof:

Let $\delta$ be the base model transition, $\delta'$ the lumped model transition. We must first show that $\delta' \circ h = h \circ \delta$. Given $<s_1, s_2, \ldots> \times <r_1, r_2, \ldots>$ by hypothesis, if $r_{1,1}, r_{1,2}, r_{1,3}, \ldots$ are the seeds corresponding to pseudo state 1,

$$\lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} I_{A_1,1}(x) \left[ \begin{array}{c} r_1 \\ i \end{array} \right] = P[A_1,1(x)]$$

That is, the proportion of neurons in pseudo state 1 that go to 1 at the next time step is just $P[A_1,1(x)]$. Similarly, the proportion in pseudo state $k$ that go to pseudo state 1 is $P[A_k,1(x)]$. Consequently, the proportion in pseudo state 1 at the next step is well defined and is just

$$\sum_{k=1}^{n} q_k \cdot P[A_k,1(x)] = q'_1$$

where $q'_1$ represents the proportion in pseudo state 1 at time step $t+1$. But, this is just what the first component of the lumped model is at $t+1$ under the lumped model transition function. That is, the first component is just

$$\sum_{k=1}^{n} q_k P_k,1(x) = \sum_{k=1}^{n} q_k P[A_k,1(x)] = q'_1$$

The other components are similarly checked. Since the output function $\lambda$ is just the proportions in the pseudo states, and $\lambda'$ gives just the pseudo state, trivially $\lambda = \lambda' \circ h$. Also, all the seeds $<r_1, r_2, \ldots>$ can be mapped to 0, and thus $h$ is a homomorphism that splits into two parts $h=k_1 \times k_2$, $k_1: S^\infty \to S'$ and $k_2: R^\infty \to R'$. By the previous lemma, the
hypothesis on the seeds is preserved at the next time step, thus completing the proof.

The proof of the infinite ideal case base model was quite easy. We went into detail only because the same techniques can be used to discuss the case of a finite number of components per block. We continue to assume that the input to each component is the same, for example, if each component gets input from every other component. However, now we will be able to exhibit a lumped model that is probabilistic and is a \(p\)-homomorphic image of a finite component base model.

4.3 Finite Number of Components in the Base Model

We begin with a base model that is a finite cross-product of ergodic machines and reduce to an appropriate lumped model in steps. The first formal lumped model turns out to be a generalized ergodic machine. The permutations of this generalized machine depend on the state of the ergodic machine.

Thus, let the base model be represented by \(M = \times_{i=1}^{N} M_i\) where a typical state is \(\langle s_1, \ldots, s_N \rangle \times \langle r_1, \ldots, r_N \rangle\). We assume the input space \(X\) is some bounded set as in the finite base model case. The transition function is of course a componentwise operation. We shall assume that the transformation \(\xi\) is a weakly mixing, faithful, permutation independent transformation. By the results of Chapter II, we know that, in fact, a multiplicative generator can be found that will satisfy these requirements. [See Theorems 2.7.1, 2.10.2, 2.10.3, Def. 4.1.3.] The output function \(\lambda\) will simply calculate the proportion of components in each of the component pseudo states.

We proceed to define a lumped model \(M\).
(a) \( X' = X \) as in the base model

(b) \( S' = R^n \) where \( R^n \) is the set of rationals (note that \( S' \) is countable. \( n \) is the number of component pseudo states).

(c) \( \Pi' = \Pi \) as in the base model

(d) \( Y' = S' \) and \( \lambda' \) is just the pseudo state output

(e) \( \zeta' = \zeta \)

Actually, we wish to define a lumped model that has state set \([S' \times R'] \mod E\) where \( E \) is an equivalence relation. A typical state representative will be \( <q_1, \ldots, q_n> \times <r'_1, \ldots, r'_N> \) modulo an equivalence relation that we proceed to characterize. First, it will always be the case that \( Nq_1 \) is an integer since \( q_1 \) represents the proportion of components in pseudo state \( i \). Now, the equivalence relation is such that a permutation of the first \( Nq_1 \) seeds gives us the same equivalence class as well as a permutation of the next \( Nq_2 \) seeds, etc. There is clearly no problem in defining the probabilities on the quotient space.

Before completing the definition of the lumped model, we attempt to give some insight into our simplification. In the base model, since we know that seed \( r_1 \) is associated with pseudo state \( s_1 \) we know which seeds correspond to the various component pseudo states. In the lumped model, however, we intend to keep only the information \( <q_1, \ldots, q_n> \) which represents the proportions in the component pseudo states. Now, since we are keeping all the original seeds, we will want to reorder them so we know the set of seeds that correspond to the different component pseudo states. In fact, we shall order them so that the first \( Nq_1 \) correspond to component pseudo state \( 1 \), the second \( Nq_2 \) correspond to component pseudo state \( 2 \), etc. However, once we know the group that corresponds, we shall be indifferent to the ordering of the seeds within
the group. Thus, we have the equivalence relation.

Rather than complete the definition of the lumped model, we first define a mapping \( h \) from the base state space to the lumped state space. We then will complete the definition of the lumped model transition in a way that will clearly show that \( h \) is in fact a homomorphism.

**Definition 4.3.1**

Given the base state space \( \langle s_1, \ldots, s_N \rangle \times \langle r_1, \ldots, r_N \rangle \) a seed \( r_i \) is a class \( k \) seed if \( s_i \) is in component pseudo state \( k \).

**Definition 4.3.2**

Let the mapping \( h \) be as follows.

\[
h(\langle s_1, \ldots, s_N \rangle \times \langle r_1, \ldots, r_N \rangle) = (\langle q_1, \ldots, q_N \rangle \times \langle r'_1, \ldots, r'_N \rangle)
\]

where

\[
q_i = \frac{1}{N} \sum_{j=1}^{N} I_i(s_j)
\]

and \( \langle r'_1, \ldots, r'_N \rangle \) is a permutation (reordering) of \( \langle r_1, \ldots, r_N \rangle \) such that \( r'_1, \ldots, r'_{Nq_1} \) are the first class 1 seeds with \( r'_1 \) being the first class 1 seed in \( \langle r_1, \ldots, r_N \rangle \) and \( r'_2 \) being the second, etc. Also, \( r'_{Nq_1+1}, \ldots, r'_{Nq_1+Nq_2} \) are the next \( Nq_2 \) class 2 seeds. We proceed similarly with the last \( Nq_n \) seeds in \( \langle r'_1, \ldots, r'_N \rangle \) being class \( n \) seeds. The mapping is clearly well defined.

We now complete the definition of the lumped model. We first describe the transition on the pseudo state part \( \langle q_1, \ldots, q_n \rangle \) and then on the seed part \( \langle r'_1, \ldots, r'_N \rangle \).

Recall that the infinite case base model used a matrix \( [p_{ij}] \) for the transition where \( P[A_{i,j}(x)] = p_{ij} \) and thus \( p_{ij} \) was the proportion of components that one expected to go from pseudo state \( i \) to pseudo state \( j \).
In the finite case, we can actually find out how many components do this. Clearly, since we know which set are class $i$ seeds, that is, the $i$th group of $<r'_1,\ldots,r'_N>$, it is an easy matter to check how many of these fall in the set $A_{i,j}(x)$. Remember that $A_{i,j}(x)$ is a well-defined set and is known, a priori.

**Definition 4.3.3**

Let the lumped model state at time $t$ be $q(t) \times r'(t) = <q_1(t),\ldots,q_n(t)>, <r'_1(t),\ldots,r'_N(t)>$. Suppose the input is $x(t)$. Let

$$q_{i,j}[x(t)] = \frac{1}{Nq_i(t)} \sum_{L \in \text{class } i} I_{A_{i,j}(x(t))}[r'_L]$$

We let $Q(t) = [q_{i,j}[x(t)]]$

Then define the next pseudo state $q(t+1)$ to be

$$q(t+1) = q(t) \cdot Q(t)$$

Notice that if the initial seeds are chosen i.i.d. then

$$\lim_{N \to \infty} q_{i,j} = p_{i,j} \quad \text{a.e.}$$

where $p_{i,j}$ is the ideal base model matrix entry.

In the above definition, if $q_i(t) = 0$ then $q_{i,j}[x(t)]$ is not well defined. In this case, it is easy to check that we can define $q_{i,j}[x(t)] = 0$ since from the transition function, any fixed value will result in the same $q(t+1)$.

It remains to define the transformation of the seed space. Given the seed vector $r'(t) = <r'_1(t),\ldots,r'_N(t)>$, we wish to define $r'(t+1)$. Ordinarily, we would simply apply the transformation $\zeta$ componentwise. However, in this case, we wish to make sure that at the next time step $t+1$, the first $Nq_1(t+1)$ seeds correspond to class 1 seeds, the next $Nq_2(t+1)$ correspond to class 2, etc. The way to do this simply involves permuting the seeds before applying the transformation $\zeta$. 

Definition 4.3.4

A seed \( r \) is a class \( m \) to \( n \) seed if it is a class \( m \) seed and
\[ I_{m,n}(x)[r] = 1. \] (Notice that a class \( m \) to \( n \) seed should actually be
a function of the input \( x \); we assume that the definition defines when
a seed \( r(t) \) is a class \( m \) to \( n \) seed.)

The above definition simply says that at the next time step,
the pseudo state corresponding to seed \( r \) will be in component pseudo
state \( n \).

Definition 4.3.5

Let the lumped model state at time \( t \) be \( q(t) \times r'(t) = \langle q_1(t),
...,q_n(t) \rangle \times \langle r'_1(t),...,r'_N(t) \rangle \). From the vector \( r'(t) \), choose class 1
to 1 seeds and place them at the head of a list. Next choose class 2
to 1 seeds and place them after the class 1 to 1 seeds. Continue this
way until class \( n \) to 1 seeds have been put on the list. Next, choose
class 1 to 2 seeds, class 2 to 2 seeds, ..., class \( n \) to 2 seeds, ..., class \( n \) to \( n \) seeds. Apply \( \zeta \) to this last to get \( r'(t+1) \).

Definition 4.3.3 and 4.3.5 constitute the transition function
of the lumped model. Notice that since we are applying a permutation
\( \pi(t) \) at each time step to the seed vector, the lumped model is a gen-
eralized ergodic machine. The following simple example in the neural
network case should clarify any source of confusion.

Example 4.3.1

In the neural network case, there were seven recovery states
0,...,6 and a neuron either went to recovery state 0 (fired) or it
dropped to the next recovery state. Suppose we have a vector \( [r'_1,...,r'_N] \).
Suppose \( r'_5 \) and \( r'_6 \) are class 0 seeds that fire and \( r'_{11} \) is a class 3 seed
that fires and no other seeds fire. Then, since the remaining seeds
drop to the next recovery state, the permutation gives us
\[[r_5', r_6', r_{11}', r_1', \ldots, r_d', \ldots, r_7', \ldots, r_{10}', r_{12}', \ldots, r_N']\]. Finally we apply \(\zeta\) componentwise to get the seed vector at the next time step of \[[\zeta(r_5'), \zeta(r_6'),
\zeta(r_{11}'), \ldots, \zeta(r_N')]\].

We now show that the mapping \(h\) of Definition 4.3.2 is in fact a
homomorphism from the base model to the lumped model.

**Theorem 4.3.1**

Let the base model state be \(s(t) < r(t) = <s_1(t), \ldots, s_N(t) > \times
<r_1(t), \ldots, r_N(t) >\) and the corresponding lumped model state be \(q(t) \times
r'(t) = <q_1(t), \ldots, q_n(t) > \times <r_1'(t), \ldots, r_N'(t) >\) under the mapping \(h\) as in
Definition 4.3.2. Then, \(h\) is a homomorphism.

**Proof:**

Given the input \(x(t)\), we must show that \(\delta' \circ h = h \circ \delta\) and \(\lambda = \lambda' \circ h\).

Consider \(\delta'[h[s(t) \times r(t)], x(t)] = q(t+1) \times r'(t+1)\). We have,

\[
q_i(t+1) = \sum_k q_k(t) q_{k,i}(x(t)) \quad \text{(Def. 4.3.3)}
\]

\[
= \sum_k q_k(t) \frac{1}{N q_k(t)} \sum_{\ell \in \text{class } k} I_{A_{k,i}}(x(t))[r_{\ell}(t)]
\]

\[
= \frac{1}{N} \sum_k \sum_{\ell \in \text{class } k} I_{A_{k,i}}(x(t))[r_{\ell}(t)]
\]

\[
= \frac{1}{N} \cdot \text{(number class } k \text{ to } i \text{ seeds at time } t)
\]

\[
= \frac{1}{N} \cdot \text{(number class } i \text{ seeds at time } t+1)
\]

\[
= \frac{1}{N} \sum_j I_{i,j}(s_j(t+1))
\]

Thus, we see that by Definition 4.3.2, \(h\) is a homomorphism on the pseudo
state space. We now check that \(h\) is a homomorphism for the seed vector.

Consider \(r(t+1)\). Some elements of this vector are class \(i\) seeds. We
wish to ascertain that in \( h[r(t+1)] \) these seeds appear as the \( i \)th group. Now, let \( r_{i1}(t+1), \ldots, r_{iL}(t+1) \) be the class \( i \) seeds. Under the base model transition this means that \( r_{i1}(t), \ldots, r_{iL}(t) \) is a union of class \( j \) to \( i \) seeds where \( j \) ranges over all component pseudo states. Now, \( r'(t) \) is just \( r(t) \) permuted, so these \( r_{i1}(t), \ldots, r_{iL}(t) \) appear in \( r'(t) \) also although possibly reordered. By the lumped model transition, however, this set of \( \bigcup_j \) (class \( j \) to \( i \) seeds) is put in the \( i \)th group at the next time step. [See Def. 4.3.5] Thus, the same seeds appear in the \( i \)th group although possibly permuted. Since the lumped model has an equivalence relation that identifies such permutation classes, we see that \( h \) is a homomorphism on the seed space. Finally, it can immediately be checked that \( h \) preserves the output function. Thus, \( h \) is, in fact, a homomorphism.

Notice that although we have shown that \( h \) is a homomorphism, we cannot in this case factor \( h \) as \( h = k_1 \times k_2 \) since we need to know the pseudo state space to decide how the seeds are to be mapped.

**Example 4.3.2**

Consider a neural network example with four recovery states 0, 1, 2, and 3. Let the base model have six neurons with the state at time \( t \) to be:

\[
[0, 1, 2, 0, 0, 2] \times [r_1, r_2, r_3, r_4, r_5, r_6]
\]

The corresponding lumped model state is clearly

\[
[1/2, 1/6, 1/3, 0] \times [r_1, r_4, r_5, r_2, r_3, r_6]
\]

Suppose that the input causes neurons 2, 4, and 6 to fire (go to state 0) in the base model. Then, the next state of the base model is

\[
[1, 0, 3, 0, 1, 0] \times \zeta[r_1, r_2, r_3, r_4, r_5, r_6]
\]
Under the mapping $h$, the lumped model state that corresponds to this state is

$$[1/2, 1/3, 0, 1/6] \times \zeta[r_2, r_4, r_6, r_1, r_5, r_3]$$

Under the lumped model transition, however, we get

$$[1/2, 1/3, 0, 1/6] \times \zeta[r_4, r_2, r_6, r_1, r_5, r_3]$$

We see that $h$ is a homomorphism because of the equivalence relation we have defined on the seed space. Notice that there is in fact a permutation of the seed space.

Call the base model and lumped model that we have so far considered $B$ and $L$ respectively. It is clear that the complexity of $L$ has hardly been reduced. Part of the problem is that we are required to permute the seed list before applying the transformation $\zeta$. Clearly, this is something that intuitively we feel we need not do, since the seeds should not be affected by permutations. However, suppose we consider a lumped model $L'$ for which we don't permute but only apply $\zeta$ component-wise?

**Theorem 4.3.2**

Let $L'$ be the lumped model that uses definition 4.3.3 for the transition on the pseudo state space and for a seed vector $r(t)$, the transition is simply $\zeta(r(t)) = r(t+1) = \langle \zeta(r_1(t)), \zeta(r_2(t)), \ldots, \zeta(r_N(t)) \rangle$. That is, the transition is a componentwise operation of $\zeta$ without permutations. Then, if $B$ is the base model as above, $L'$ is a probabilistic homomorphic image of $B$.

**Proof:**

It is clear from the definitions that both $B$ and $L'$ are ergodic machines. Furthermore, the representative class of $L'$ is the same as
the representative class of \( L \). This follows since the difference between \( L \) and \( L' \) is only that one has a set of time indexed permutations \( \{\pi(t)\} \) operating at each step. Since \( \pi \) is assumed to be permutation independent, both faithfully represent the associated automaton. We have shown in Theorem 4.3.1 that there exists a homomorphism \( h \) from \( B \) to \( L \). Consequently, \( h \) is a \( p \)-homomorphism from \( B \) to \( L' \). (See Definitions 4.1.3-4.1.5)

We now consider a further simplification. So far, we have a lumped model \( L' \) that uses \( N \) seeds where \( N \) is the number of components. We shall see that we can find a lumped model \( L'' \) that is in the same representative class as \( L \) but that uses at most \( n^2 \) seeds where \( n \) is the number of component pseudo states.

To make things easier we recall the transition function of \( L' \). A typical state is \( <q_1, \ldots, q_n> \times <r_1, \ldots, r_N> \) and the transition function is defined as follows:

The new pseudo state vector is:

\[
q(t+1) = q(t) \cdot Q(t)
\]

where the matrix \( Q \) is determined by

\[
q_{i,j}(t) = \sum_{L \in \text{class}} I_{A_{i,j}}(x(t)) \frac{r_L}{Nq_i(t)}
\]

The new seed vector is simply \( <\zeta(r_1), \ldots, (r_N)> \). The complexity of this model is due to the fact that we are using \( N \) seeds operating in parallel, plus the fact that we must actually determine to which class a seed transitions. We seek a model that also faithfully represents the same associated automaton as \( L' \) does but that has fewer seeds in general.

To clarify the derivation we first consider an extension of the
lumped model L' by including a redundant matrix as part of the state space. Let \( K_{i,j}(t) = \sum_{\xi} \sum_{\text{class } i} I_{A_{i,j}(x(t))}[r_{\xi}] \). Define the matrix \( K(t) \) to be \([K_{1,1}(t)\]) . Then, we can define the extended state space as

\[
<q_1, \ldots, q_n> \times K \times <r_1, \ldots, r_N>
\]

Using this information, it is easy to see that the matrix \( Q(t) \) can be defined as

\[
\begin{bmatrix}
\frac{K_{1,1}}{Nq_1} & \frac{K_{1,2}}{Nq_1} & \cdots & \frac{K_{1,n}}{Nq_1} \\
\frac{K_{2,1}}{Nq_2} & \frac{K_{2,2}}{Nq_2} & \cdots & \frac{K_{2,n}}{Nq_2} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{K_{n,1}}{Nq_n} & \frac{K_{n,2}}{Nq_n} & \cdots & \frac{K_{n,n}}{Nq_n}
\end{bmatrix}
\]

Furthermore, it is easy to see that

\[
q_{i}(t+1) = \frac{1}{N} \sum_{j=1}^{n} K_{j,i}(t)
\]

Note that \( K_{i,j} \) clearly depends on both the vector \(<q_1, \ldots, q_n>\) as well as \(<r_1, \ldots, r_N>\).

The interesting point is that we can view the method of calculating \( K_{i,j} \) as simply one way of implementing the calculation of a random variable with a specific distribution. In fact, it is apparent that since the transition seed space \( A_{i,j}(x(t)) \) has a well defined probability, say \( p_{i,j}(t) \), we can view the calculation of the vector \( K_{i,1}, K_{i,2}, \ldots, K_{i,n} \) as defining a multinomial distribution. That is,

\[
\text{Prob}\{k_{i,j} = a_j \text{ for } j=1, \ldots, n\} = \frac{(Nq_{i,j})!}{a_1! \cdots a_n!} p_{i,j}^a \cdots p_{i,n}^a
\]

The reason
this is true is that $L'$ was represented faithfully so this is the only information we retain about the pseudo states and seeds since we originally chose the initial seeds i.i.d. Since the vectors $\langle K_{i,1}, \ldots, K_{i,n} \rangle$ are generated multinomially, there is no reason not to use some other method for the calculation if this is convenient, as long as the probabilities of the $K_{i,j}$ remain the same. Thus, we have shown that we can use the $n^2 K_{i,j}$ values instead of the original $N$ seeds in our calculations. Notice that all the $K_{i,j}$ are not independent and thus we would use fewer than $n^2$ values in general. The following example should clarify the situation.

**Example 4.3.3**

In the neural network model we see that the matrix $K$ is simply

\[
\begin{bmatrix}
\frac{K_0}{Nq_0} & 1 - \frac{K_0}{Nq_0} & \cdots & \cdots & 0 \\
\frac{K_1}{Nq_1} & 0 & 1 - \frac{K_1}{Nq_0} & \cdots & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
\frac{K_6}{Nq_6} & 0 & \cdots & \cdots & 1 - \frac{K_6}{Nq_6}
\end{bmatrix}
\]

We also have

\[
q_0(t+1) = \frac{1}{N} \sum_{i=0}^{6} K_i \\
q_1(t+1) = q_0(t) - \frac{K_0}{N} \\
\vdots \\
q_5(t+1) = q_4(t) - \frac{K_4}{N}
\]
Clearly, we need only generate the seven seed values \(<K_0,\ldots, K_6>\). Furthermore, \(K_i\) is Binomial \((Nq_i, p_{i0})\) where \(p_{i0} = F(x - 27e^{-i})\) and \(F\) is Normal \((0, 10)\). \(K_i\) being Binomial \((Nq_i, p_{i0})\) means that

\[
\text{Prob}[K_i = j] = \binom{Nq_i}{j} p_{i0}^j (1 - p_{i0})^{Nq_i-j}
\]

Notice that \(p_{i0}\) is simply the theoretical entry in the transition matrix for the ideal infinite number of neurons case. We can in fact consider the count \(K_i\) as using a rejection method for generating the value of a random variable that is distributed Binomial\((Nq_i, p_{i0})\). Recall that a rejection method for calculating a binomially distributed, Binomial\((N,p)\) random variable \(X\) says to take \(N\) uniformly distributed random variables between \([0,1]\) and count the number that are less than or equal to \(p\).

In this example, we have found a lumped model \(L''\) that represents the same associated automaton that \(L'\) does. Since we can find a faithful transformation \(\tau''\) on \(L''\), we see that \(L'\) and \(L''\) are in the same representation class. Notice the simplicity of the final lumped model that we have obtained. Given a pseudo state \(<q_0,\ldots, q_6>\), the next values of the pseudo state are

\[
\frac{1}{N} \left< \sum_{i=0}^6 K_i Nq_0 - K_0, \ldots, Nq_4 - K_4, Nq_5 + Nq_6 - K_5 - K_6 \right>
\]

It can also be shown that the expected value of this next pseudo state is exactly the next pseudo state of the lumped model corresponding to the ideal base model case.

The exposition of this section and this last example have made clear that in fact, there exists a probabilistic homomorphism from \(B\)
to $L''$. We consequently have the next theorem.

**Theorem 4.3.3**

Given the base model $B$, there exists a lumped model $L''$ that is a probabilistic homomorphic image of $B$. $L''$ has at most $n^2$ seeds in parallel in the seed space vector, where $n$ is the number of component states.

**Proof:**

The proof was in the derivation of this section. For an appropriate $\xi''$, $L''$ is a faithful ergodic machine in the same representation class as $L'$. Since a p-homomorphism from $B$ to $L'$ exists (Theorem 4.3.2) a p-homomorphism from $B$ to $L''$ also exists.

Our simplification has yielded a lumped model that can be implemented on the computer. Since, for the neuron example, we need only seven seeds operating in parallel, the model is much simpler than the original base model. Since the model as implemented is really the corresponding probabilistic automaton, we call it the probabilistic lumped model.

**4.4 A Probabilistic Lumped Model**

In the previous section (Example 4.3.3), we derived a probabilistic lumped model that should exhibit the same trajectory behavior as the base model of Chapter III. The probabilistic lumped model of course assumes that each neuron receives identical input and consequently, the model is more accurate wherever the number of neighbors per neuron approaches the number of neurons in a block. In fact, whenever we consider a base model such as 50 neurons and 50 neighbors per neuron in the single block studies, the probabilistic lumped model exactly models
the base model.

In order to compare base model and probabilistic lumped model runs, we expanded the simulation package of Chapter III. In the actual computer implementation, it turned out to be easier to in fact use a rejection method to generate the binomial distributions. However, when generating Binomial\(N,p\) for large values of \(N\), it is possible to use the normal approximation

\[
K = Z \sqrt{Np(1-p)} + Np
\]

where \(Z\) is standard normal \(N(0,1)\). This was done to speed up the computations of the binomially distributed values.

The first consideration of interest was to examine the behavior of the probabilistic lumped model when there was Positive Feedback and we were considering a single block. The comparison is with the base model runs of section 3.5 and, in particular, comparisons with figs. 3.5.5-3.5.12. In those figures, we saw how the base model less frequently deviated along the "force field" as the number of neurons and number of neighbors increased. Figs. 4.4.1-4.4.4 are probabilistic lumped model runs of 50, 100, 200, and 400 neurons respectively. Because of the assumption of identical input, these correspond really to 50 neurons 50 neighbors, 100 neurons 100 neighbors, etc. Of course, once the base model has a sufficient number of neighbors, the identical input assumption is fairly good anyways. We also made two additional base model runs for additional comparisons, and these are shown in figs. 4.4.5 and 4.4.6. The similarity between the base model and probabilistic lumped model is quite striking, especially comparing fig. 3.5.12 with fig. 4.4.1 and comparing fig. 4.4.2 with fig. 4.4.5.

The next situation of interest was to consider the flip-flop
Figure 4.4.1. Probabilistic Lumped Model Run 50 Neurons (2 runs)
Figure 4.4.2. Probabilistic Lumped Model Run 100 Neurons
Figure 4.4.3. Probabilistic Lumped Model Run 200 Neurons
Figure 4.4.4. Probabilistic Lumped Model Run 400 Neurons
Figure 4.4.5. Base Model Run 100 Neurons 100 Neighbors
Figure 4.4.6. Base Model Run 200 Neurons 100 Neighbors
example of section 3.6. In that section we considered three experiments with the flip-flop: (1) Setting the Flip-Flop from Lockstep, (2) Stability of the Set State, and (3) Resetting the Flip-Flop.

The behavior of the probabilistic lumped model in setting from the lockstep mode was as predicted. An input level of 50 would set the flip-flop whereas an input level of 40 would not. An input level of 45 also would not set it in general although occasionally it might. This is the same behavior that our tests with the base model with 50 neurons per block and 50 or 40 neighbors per neuron in the other block resulted in. For the probabilistic lumped model, we used 50 neurons per block.

In Chapter III, table 3.6.1 gives the information on the stability of the base model flip-flop state. Table 4.4.1 gives the results of the two different lumped model experiments done for this case. It also includes the base model run as well as an additional base model run for comparison.

Finally, we consider the case of resetting the flip-flop. Table 3.6.3 shows the reset behavior of the base model. The corresponding table for the case of the probabilistic lumped model behavior is table 4.4.2. Since there are not really enough probabilistic lumped model runs, we see that the correspondence is fairly good.

The statistics of this section have shown that the probabilistic lumped model does reflect base model behavior as it should. However, the statistics also point out the importance of a theoretical derivation of the lumped model correspondence, since, for small numbers of runs, particular statistics could be misleading.
TABLE 4.4.1

PROBABILISTIC LUMPED STABILITY VS. BASE MODEL STABILITY

<table>
<thead>
<tr>
<th>Block Size</th>
<th>Number of Neurons</th>
<th>Number of Runs that Unset After x-y Time Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-4</td>
<td>5-8</td>
</tr>
<tr>
<td>50</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Base Model

<table>
<thead>
<tr>
<th>Block Size</th>
<th>Number of Neurons</th>
<th>Number of Neighbors</th>
<th>Number of Runs that Unset After x-y Time Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-4</td>
<td>5-8</td>
<td>9-12</td>
</tr>
<tr>
<td>50</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>20</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>50</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>500</td>
<td>20</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>500</td>
<td>50</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
### Table 4.4.2

**Probabilistic Lumped Model**

<table>
<thead>
<tr>
<th>Block Size</th>
<th>Input = 25</th>
<th>Input = 30</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reset</td>
<td>Lockstep</td>
</tr>
<tr>
<td>50</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>100</td>
<td>23</td>
<td>6</td>
</tr>
<tr>
<td>500</td>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>

**Base Model**

<table>
<thead>
<tr>
<th>Block Size</th>
<th>Input = 25</th>
<th>Input = 30</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reset</td>
<td>Lockstep</td>
</tr>
<tr>
<td>50</td>
<td>37</td>
<td>23</td>
</tr>
<tr>
<td>50</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>21</td>
</tr>
<tr>
<td>100</td>
<td>47</td>
<td>18</td>
</tr>
<tr>
<td>100</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>100</td>
<td>18</td>
<td>8</td>
</tr>
<tr>
<td>500</td>
<td>31</td>
<td>0</td>
</tr>
</tbody>
</table>
4.5 Finite Number of Neighbors

We now turn to a consideration of the situation that obtains when each component of a block of the base model cannot be expected to get the identical value of input. That is, we are considering the case when we have a connection pattern such that the different base model components get differing input. We shall strive for understandability of the simplification ideas in this section and will not attempt to be totally precise. Again, we examine the case of an aggregation to a single block and consider the input to be from this block also. The generalization to the many block case is not difficult.

Since each component will get input as a function of its neighborhood set, we shall explicitly represent the connection pattern. First, however, we represent the state space of our base model. Suppose $N$ components are in the base model. Then, the state of the base model can be represented by $\chi(s_i, r_i)$ where $s_i$ is the component state and $r_i$ is the noise as we have previously described these values. The input can be determined knowing the function of the state that defines the input and knowing the connection pattern. However, we can be redundant and include it explicitly as part of the state set. Thus, the state will be $\chi(s_i, r_i, x_i)$ where the vector $<x_1, \ldots, x_N>$ will be called the input vector. For simplicity, we only consider the situation in which certain component states can be thought of as emitting a pulse and the input to a component is the sum of the connection weights of the neighbors that are emitting a pulse. This corresponds to considering the input to a neuron in a neural net example as being determined by the sum of synapse weights of its neighbors that are firing.
Definition 4.5.1

Given the state $\prod_{i=1}^{N}(s_i, r_i, x_i)$ and thus the input vector $x = \langle x_1, \ldots, x_N \rangle$, a vector $x'$ is a strong permutation of $x$ if it is a permutation of $x$ such that values corresponding to any fixed component state are also permuted.

Example 4.5.1

Consider an example with two component states 0 and 1, and 5 components in the base model. Let the pseudo state vector be $\langle 0, 1, 0, 0, i \rangle$ with input values $x = \langle 5, 10, 15, 20, 25 \rangle$ (notice that we have both $5$s and emitting a pulse and varying connection weights). Then, $x' = \langle 20, 25, 15, 5, 10 \rangle$ would be a strong permutation since the subvectors $\langle 5, 15, 20 \rangle$ and $\langle 10, 25 \rangle$ are permuted. However $\langle 10, 5, 15, 20, 25 \rangle$ would not be a strong permutation.

The idea of a strong permutation is that the distribution of values corresponding to each component state remains the same under a strong permutation. Now if we generate a strong permutation of $x$, rather than $x$ itself, we can claim that the lumped model percent in state $i$ vector is the same for both inputs.

Proposition 4.5.1

Let $B$ be an ergodic machine with state $\prod_{i=1}^{N}(s_i, r_i, x_i)$ and suppose at time step $t_0$, instead of using $x(t_0)$ we use the input vector $x'(t_0) = \pi(t_0) \circ x(t_0)$ where $\pi$ is a strong permutation. Then, there is a generalized ergodic machine $B'$ that uses $x'$ at time $t_0$ and that has identical percent in state vector through time step $t_0+1$.

Proof:

$B'$ is just $B$ through time step $t_0-1$. At $t_0$, apply the
permutation $\pi(t_0)$ to the seed vector $<r_1, \ldots, r_N>$. Now, because $x'$ is a strong permutation, corresponding to any component state $i$, we have identical seeds and distribution of inputs, and thus, the next percent in state vector is the same for $B$ and $B'$.

It is important to note that the above works for one time step only. The problem at the next time step is that although we have the same percent in state vector, the pseudo state vector has been permuted. Since the connection pattern is fixed, that is, the input wires are the same and do not change, we have no idea of how the input vector corresponding to the permuted pseudo state compares with the input vector corresponding to the original pseudo state. We shall see that if we hypothesize that regardless of the permutation of the pseudo state, the input vector has an identical distribution in corresponding component states, then we can effect a simplification that is preserved over all time steps. First however we investigate the explicit representation of the connection pattern.

We shall find it convenient to represent the connection pattern as a matrix $M$ where $M = [m_{ij}]$ is an $N \times N$ matrix and $m_{ij}$ represents the connection strength from the $j$th component to the $i$th component. Given the pseudo state vector $s = <s_1, \ldots, s_N>$ let $f$ be the mapping that determines whether or not $s_i$ is emitting a pulse or not and if it is we let $f(s_i) = 1$ otherwise $f(s_i) = 0$. It is clear that the vector $f(s)$ is a vector composed of 1's and 0's and has length $N$. Furthermore, if $x$ is the input vector, we see that we can represent $x$ as follows:

$$x = M \cdot f(s)$$

**Example 4.5.2**

Consider a neuron net example with recovery states 0, 1, and 2
and let $0$ be the only firing state. Suppose there are five neurons in
the base model with the neighbors of neuron one being four and five, the
neighbors of neuron three being one and no other connections. Suppose
the synapse weights are identically $3$. Then, the connection matrix is

$$
M = \begin{bmatrix}
0 & 0 & 0 & 3 & 3 \\
0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Consider a typical pseudo state $s = <1,0,2,0,2>$. Then $f(s) = <0,1,0,1,0>$
and the input vector $x$ is just $<3,0,0,0,0>$.

We are now ready to give a sufficient condition for a lumped
model that is a p-homomorphic image of the base model. Essentially,
what we need is that the connection matrix $M$ be such that a permutation
of the pseudo state vector $\pi_1(s)$ should give an input vector that is a
strong permutation of the input vector $\pi_1(x)$ for all permutations $\pi_1$.

**Proposition 4.5.2**

Suppose that for every permutation $\pi_1$, there exists a strong
permutation $\pi_2$ such that the following diagram commutes for every pseudo
state $s$.

Then there exists a p-homomorphic lumped model with the pseudo
state as the percent in state vector.
Proof:

We have really provided sufficient conditions for the generalization of Proposition 4.5.1. Since at each time step, knowing the percent in state vector, we can discover the distribution of the inputs, we can appropriately permute the seeds for a homomorphism. The hypotheses of this proposition are the sufficient conditions for the input distributions to be determined by the percent in state vector only, since we can in fact choose any pseudo state vector that has the same percent in state values and apply it to M to get the inputs.

The formalism of the above should not obscure the essential simplicity of the concepts. We want to be able to say that after one time step, as in Proposition 4.5.1, even if we have a permuted pseudo state vector, the input distributions are still the same for each component state. Thus, we need only retain the percent in state information to calculate these input values. Much of the rest of our discussion will be an attempt to show under what conditions the matrix M can satisfy the hypotheses of Proposition 4.5.2.

Example 4.5.3

Consider a connection pattern such that each neuron is connected to every other neuron but is not connected to itself and the connection weight is identically a. Then, the resulting matrix M for the neural network examples satisfies Proposition 4.5.2 as can easily be checked. Notice that in this case, although the values of the input are identical for all neurons in a particular component state, for different component states the values may not be the same. We show this for a simple case. Suppose we have recovery states 0,1 and 2 and we consider four neurons.
Let the pseudo state be \( s = <0,0,1,2> \). Then \( f(s) = <1,1,0,0> \). Since
\[
M = \begin{bmatrix}
0 & a & a & a \\
a & 0 & a & a \\
a & a & 0 & a \\
a & a & a & 0
\end{bmatrix}
\]
we see that \( M \cdot f(s) = x = <a,a,2a,2a> \). Similarly, if we consider the
permutation \( \pi_1 \) of the state set to \( <1,2,0,0> \) we have \( M \cdot f(\pi_1(s)) = <2a,2a, a,a> \). Since \( \pi_1 \circ x = M \cdot f(\pi_1(s)) \) we see that the strong permutation for the
state \( s \) and permutation \( \pi_1 \) is simply \( \pi_2 = \text{identity} \).

Notice that for the lumped model, all that we would need is the
percent in state vector and the matrix \( M \). The matrix \( M \) is important
since it allows us to calculate the distribution of the input values.

Although the hypotheses seem strong, we shall see that they
actually obtain in the limiting case of an infinite number of components
as long as the connection matrix is sufficiently "random". We of course
continue to have only a finite number of neighbors per component.

We shall now consider the following matrix as an example of a
"random" connection matrix. Suppose there are three neighbors per com-
ponent. Let the connection weight be identically \( a \). Consider the fol-
lowing infinite matrix.
\[
M_{\infty} = \begin{bmatrix}
0 & a & a & a & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots \\
0 & 0 & 0 & 0 & a & a & a & 0 & 0 & 0 & 0 & 0 & \ldots \\
0 & 0 & 0 & 0 & 0 & 0 & a & a & a & 0 & 0 & 0 & \ldots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a & a & a & \ldots \\
& & & & & & & & & & & & \ldots \\
& & & & & & & & & & & & \ldots \\
& & & & & & & & & & & & \ldots \\
\end{bmatrix}
\]

Essentially, each row, which represents the connection to other neurons,
takes input from neurons that are the next three farther out. Clearly,
this form only works when the matrix is infinite. Given any pseudo
state \( s \), this determines an input \( x \) as before. Later on, we shall make some comments on the "randomness" of this matrix.

Let us attempt to intuitively clarify what we shall show more formally below. We assume that we have a base model with a large number of components and a finite number of neighbors per component. Assume that the connection pattern is fixed and is in a sense "random". Then, we would say that although we are running the base model with fixed connections, most runs can be treated as if the connections were redefined at each time step and consequently, a lumped model that ignores the connection pattern is sufficient to represent this base model for most base model runs. That is, we can imagine the sampling of neighbors at each time step, randomly and independently, as giving the same results as we get by the sampling determined by our fixed connection pattern.

There will be two steps in the argument. First, we will want to say that most initial pseudo states will give rise to the same distribution of inputs. Secondly, we want to say that the transition to the next state will result in a pseudo state that is still of this type. The distribution of inputs will of course be able to be calculated by knowing only the percentage in state vector or the distribution of the components in the various component states. We now consider the more formal arguments.

Let \( n \) be the number of component states. Let \( k \) be the number of neighbors per component. Let \( N \) be the number of components. We shall let \( X^Y \) be the function space of all mapping from \( Y \rightarrow X \). Suppose for every value \( N \), we let \( S_N \) be the set of all pseudo states corresponding to \( N \) components and we let \( M_N \) correspond to some fixed connection pattern. We wish to define a mapping \( \phi \) from the set of pseudo states \( S_N \) to
the n-dimensional function space $\left[ [0,1]^I \right]^n$ where $I$ is the set of integers.

**Definition 4.5.2**

Let $S_N$ be the set of pseudo states with a typical pseudo state $<s_1, \ldots, s_N>$ and a typical input vector $<x_1, \ldots, x_N>$ as determined by $M_N$. Then, we can define a mapping $\phi$ by $\phi: S_N \rightarrow \left[ [0,1]^I \right]^n$ where $\phi(<s_1, \ldots, s_N>) = \psi = (\psi_1, \ldots, \psi_n)$ and $\psi_i: I \rightarrow [0,1]$. $\psi_i$ calculates the percentage of input values corresponding to component state $i$ that have input values of $0,1,2,\ldots,k$. (We assume that input can be normalized to 1 and so these are the only input values.) $\psi_i$ is 0 for all other integers.

The above definition simply defines how the input values are distributed corresponding to each component state. We give an example that will clarify the definition.

**Example 4.5.4**

Let us consider a neuron net example with recovery states 0, 1, and 2. Suppose there are six neurons in the base model and three neighbors per neuron. Then, we may assume that the possible input values are $0,1,2$ or 3. Let a pseudo state $s = <0,1,2,0,0,2>$ with a corresponding input vector $x = <1,2,3,1,2,0>$. We can easily calculate the distribution $\psi_i$ for each component state.

\[
\begin{align*}
\psi_0(0) &= 0 & \psi_0(1) &= 2/3 & \psi_0(2) &= 1/3 & \psi_0(3) &= 0 \\
\psi_1(0) &= 0 & \psi_1(1) &= 0 & \psi_1(2) &= 1 & \psi_1(3) &= 0 \\
\psi_2(0) &= 1/2 & \psi_2(1) &= 0 & \psi_2(2) &= 0 & \psi_2(3) &= 1/2
\end{align*}
\]

Before showing how the concept of the $\phi$ map relates to our previous concept of strong permutations, we generalize the notion of strong permutation to the pseudo states. (See Definition 4.5.1.)
Definition 4.5.3

Two pseudo states $s$ and $s'$ are strong permutations of each other if there exists a permutation $\pi_1$ such that $s' = \pi_1 \circ s$ and there exists a strong permutation $\pi_2$ such that the diagram of Proposition 4.5.2 commutes.

The above definition simply says that the corresponding input vectors $x$ and $x'$ have the property that $\pi_1 \circ x$ is a strong permutation of $x'$.

Proposition 4.5.3

Let $s$ and $s'$ be two pseudo states that are permutations of each other. Then $s$ and $s'$ are strong permutations if and only if $\phi(s) = \phi(s')$.

Proof:

$\Rightarrow$ Let $s$ and $s'$ be strong permutations. That is $s' = \pi_1 \circ s$ and $x'$ and $\pi_1 \circ x$ are strong permutations. But, this implies that $x'$ and $\pi_1 \circ x$ have the same distribution. Thus, $s$ and $s'$ have the same input distributions in corresponding component states, or $\phi(s) = \phi(s')$.

$\Leftarrow$ Assume $\phi(s) = \phi(s')$. Since we know $s'$ is a permutation of $s$, $s' = \pi_1 \circ s$, we need only show that it is a strong permutation. But this is obvious since the two input distributions are the same for any component state, that is, $x'$ and $\pi_1 \circ x$ are strong permutations.

What we now intend to show is that almost all states will map to a particular subset of points under the $\phi$ mapping. We will call this subset of image points the ideal points.

Definition 4.5.4

Let $<q_1, \ldots, q_n>$ be a pseudo state vector of the lumped model, that is, it corresponds to a percentage in state vector of the base
model. Then \( S_N / \langle q_1, \ldots, q_n \rangle \) is the set of pseudo states of \( S_N \) that have \( \langle q_1, \ldots, q_n \rangle \) as their percent in state vector. Note that this partitions the space \( S_N \) and each partition class is a class of permutations.

We now wish to associate a specific point in the function space with the lumped pseudo state vector \( \langle q_1, \ldots, q_n \rangle \). For this purpose, we actually need the percentage of components emitting a pulse. We shall assume that the function \( f \) takes the vector \( \langle q_1, \ldots, q_n \rangle \) and calculates a value \( p \) which is the percent emitting a pulse, or the number of components firing. Then,

**Definition 4.5.5**

Let \( \langle q_1, \ldots, q_n \rangle \) be a pseudo state vector of the lumped model and let \( p \) be the percent emitting a pulse. Then, the ideal point corresponding to \( \langle q_1, \ldots, q_n \rangle \) is simply

\[
\psi_i = \binom{k}{m} p^m (1-p)^{k-m} \quad \text{for } m = 0, \ldots, k \text{ and for } i = 1, \ldots, n.
\]

Notice that \( \psi = \langle \psi_1, \ldots, \psi_n \rangle \) will have all entries the same function when \( \psi \) is the ideal point.

The ideal point simply says that we are looking at the distribution of inputs when we have an infinite number of components and truly a random connection pattern. For such a situation, the ideal point is the distribution that one would expect, knowing the percent emitting a pulse and knowing that there are \( k \) neighbors per neuron and assuming random sampling.

We now want to let the number of components approach \( \infty \) and see what happens. Suppose we let \( S_\infty \) be the set of countably infinite sequences \( \langle s_1, s_2, s_3, \ldots \rangle \) and we let \( M = \langle V_1, V_2, \ldots \rangle \) be the random matrix as before whose first few rows are
$V_1 = 0 \underbrace{1 1 1 \ldots 1}_{k \text{ times}} 0 0 0 0 \ldots$

$V_2 = 0 0 0 0 \ldots 0 \underbrace{1 1 1 \ldots 1}_{k \text{ times}} 0 0 0 0 \ldots$

$\ldots$

$V_n = [k(n-1)+1] \ 0^\prime s \text{ followed by } k \ 1^\prime s \text{ followed by } 0^\prime s.$

Notice that this is the generalization to $k$ neighbors of the matrix that we previously defined where we have normalized the input to 1.

Now, what can we mean by the class of permutations corresponding to a lumped vector $<q_1, \ldots, q_n>$? We proceed as follows. We assume that the set $S_\infty$ is generated by choosing each component state i.i.d. from the distribution $<q_1, \ldots, q_n>$. By the law of large numbers, the set of infinite sequences that can be said to have the distribution $<q_1, \ldots, q_n>$ as their percentage vector, has measure 1. That is, we have converted the set of pseudo states into a probability space. If the set of pseudo states is represented by $\Omega$, by choosing the state of each component i.i.d. from the distribution $<q_1, \ldots, q_n>$, we have imposed a probability measure $\mathbb{P}$ on subsets of $\Omega$. Notice that a point $\omega \in \Omega$ is just a pseudo state $\omega = <s_1, s_2, s_3, \ldots>.$

Since the set of pseudo states is just a probability space we can proceed to define a sequence of random variables on $\Omega$ corresponding to each component state $i$.

Definition 4.5.6

For each component state $i$, we define the sequence of random variables $W_{1}^{i}, W_{2}^{i}, W_{3}^{i}, \ldots$ on $\Omega$ as follows. Given an $\omega = <s_1, s_2, \ldots>$,
$W^i_1$ finds the first component in component state $i$, say it is $n_1$, and determines the input that corresponds by applying $V^i_{n_1}$ to the value $f(\omega)$ [see example 4.5.2]. That is, $W^i_1(\omega) = V^i_{n_1} \cdot f(\omega)$. The $V'$s are of course from the connection matrix $M$. Similarly $W^i_2$ finds the second component in component state $i$, say $n_2$ and does the same thing. In general

$$W^i_j(\omega) = V^i_{n_j} \cdot f(\omega).$$

**Lemma 4.5.1**

Let $M$ be the matrix $M = \langle V_1, V_2, \ldots \rangle$ as defined above and consider the random variables $W^i_1, W^i_2, W^i_3, \ldots$ defined in $\Omega$ as above. Then, these random variables are independent and identically distributed.

**Proof:**

It is clear that the $W^i_j$ are identically distributed since they each examine a set of $k$ neighbors or components of the pseudo state $\omega \in \Omega$. Since we assume that the component states are chosen i.i.d., it is clear that any $k$ neighbors will give the same distribution. Furthermore, if we take any subset $W^i_{j_1}, W^i_{j_2}, W^i_{j_3}, \ldots, W^i_{j_n}$ of the random variables, since they examine disjoint neighborhoods (recall the definition of $M$), they are independent.

**Theorem 4.5.1**

If we choose the pseudo state by choosing each component i.i.d. with the distribution $\langle q_1, \ldots, q_n \rangle$, and we assume the connection matrix $M = \langle V_1, V_2, \ldots \rangle$ as above, then the set of pseudo states that map to the ideal point corresponding to $\langle q_1, \ldots, q_n \rangle$ has measure 1.

**Proof:**

For the proof, we need to show that the distribution of input values corresponding to any component state for any pseudo state...
\( \omega = \langle s_1, s_2, s_3, \ldots \rangle \) is identical almost everywhere. Translated, we need only show that the values \( W^i_1(\omega), W^i_2(\omega), \ldots \) are identically distributed. But, this is not difficult. Since the random variables \( W^i_1, W^i_2, W^i_3, \ldots \) are i.i.d. we can assume that they are generated by some ergodic process [Breiman (8), p. 107]. That is,

\[
W^i_j(\omega) = W^i_1(\zeta^j(\omega)) \quad \text{for some } \zeta \text{ an ergodic transformation.}
\]

Now, by the ergodic theorem [Breiman (8), p. 111]

\[
\lim_{N \to \infty} \frac{1}{N} \sum_{j=0}^{N-1} W^i_j(\omega) = \frac{1}{N} \sum_{j=0}^{N-1} W^i_1(\zeta^j(\omega)) = \int W^i_1 dP \quad \text{a.e.}
\]

That is to say, the average of the elements \( W^i_j(\omega) \) is the same for all \( \omega \) almost surely. Similarly, the higher moments are the same which proves that for any \( \omega \), it is almost surely true that we get the same distribution of inputs. It is easy enough to check that the distribution of \( W^i_1 \) is just the ideal point. Thus, the theorem is proved.

**Definition 4.5.7**

Let the subset of permutations of \( S/\langle q_1, \ldots, q_n \rangle \) that map into the ideal point corresponding to \( \langle q_1, \ldots, q_n \rangle \) under the mapping \( \phi \) be called \textbf{proper} permutations.

**Theorem 4.5.2**

If we look at the permutation classes as \( N \to \infty \) most permutations corresponding to a lumped model pseudo state are strong permutations of each other.

**Proof:**

By the previous theorem, most permutations corresponding to a lumped model pseudo state are proper. By Proposition 4.5.3, it is clear that proper permutations are strong.
We can now use Proposition 4.5.2 to see that the lumped model is a p-homomorphic image of the base model. We need only check that the condition that we perceive the pseudo state as being generated by i.i.d. choice of component states still holds at the next time step. But this is fairly easy to see. It is apparent that the distribution generated by the input values are not only identically distributed, but can also be considered to be generated i.i.d. Thus, the next pseudo states can also be considered to be generated i.i.d. Essentially, we can look at this more precisely by saying that the ensemble of states at time step t+1 cannot have a strange distribution without there being a high probability of generating this strange distribution.

Notice that the form of the matrix depended only on the fact that the random variables $W_1^i, W_2^i, \ldots,$ were i.i.d. If we can guarantee this result using some other connection matrix M, then this matrix would satisfy our concept of being random. We see that we are basically requiring the neighborhood sets to be disjoint and to not contain the component itself as a neighbor.

For large numbers of components, we can thus confidently use the lumped model. Rather than most permutations mapping exactly to the ideal point, they would map close to the ideal point. Since our original base model has independent noise generated at each time step, the fact that they do not map exactly to the ideal point can be subsumed as part of the noise. Thus, we would expect that a probabilistic lumped model with some additional noise as generated due to the small number of neighbors would appropriately model the base model. The runs of Chapter III essentially show that nothing unexpected happens due to finite neighbors. However, to more precisely represent such a probabilistic lumped model,
one would have to go into a discussion of the topology on the function space. This concludes our discussion of the case of a finite number of neighbors.

4.6 Conclusion

This ends the discussion of applying ergodic machines to simplifications. We saw how the concept could clarify the relationship between two probabilistic systems and also how one could use the structural properties to gain an understanding of the simplification procedures. In the next chapter, we conclude with some remarks about future directions.
CHAPTER V

FUTURE DIRECTIONS

In the foregoing chapters, we have developed new methods for studying the simplification of probabilistic systems. In particular, the usefulness of the concept of an Ergodic Machine is apparent, for this provides a structural handle with which to study probabilistic models.

In studying ergodic machines, we saw the importance of the concept of a faithful transformation. We characterized the faithfulness of multiplicative generators and showed that universally faithful transformations do not exist. We made connections to pseudo random number generators and considered the concepts of permutation independence and probabilistic homomorphisms. We also showed the theoretical derivations of lumped models for (1) Infinite number of components and neighbors, (2) Infinite number of components and finite number of neighbors, and (3) Finite number of components and infinite number of neighbors. In this last case, we really mean a complete neighborhood set so that the input is the same to each component.

We experimentally studied a neuron network example for the case of finite number of components and neighbors. In this case, we called the lumped model an approximate homomorphic image. Table 5.1.1 gives a synopsis of the results of this analysis. Notice that in the case of positive feedback we cannot distinguish between aperiodic behavior and cyclic behavior, whenever we do not have a steady state. The data that

180
<table>
<thead>
<tr>
<th>Type of Model</th>
<th>Number of Neurons</th>
<th>Number of Neighbors</th>
<th>Result or Prediction</th>
<th>Method Used</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>No Connections--</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lumped</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>Steady State</td>
<td>Theoretical</td>
</tr>
<tr>
<td>Base</td>
<td>25-1000</td>
<td>0</td>
<td>Fluctuates about</td>
<td>Simulation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Steady State</td>
<td></td>
</tr>
<tr>
<td><strong>Positive Feedback--</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lumped</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>Steady State</td>
<td>Heuristic and</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Simulation</td>
</tr>
<tr>
<td>Base</td>
<td>50-1000</td>
<td>20-60</td>
<td>Aperiodic or</td>
<td>Simulation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cyclic</td>
<td></td>
</tr>
<tr>
<td>Lumped (Force Field)</td>
<td>finite</td>
<td>finite</td>
<td>Non Steady State</td>
<td>Heuristic and</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Simulation</td>
</tr>
<tr>
<td>Probabilistic Lumped</td>
<td>finite</td>
<td>$\infty$</td>
<td>Aperiodic or</td>
<td>Theoretical and</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cyclic</td>
<td>Simulation</td>
</tr>
<tr>
<td><strong>Flip-Flop--</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lumped</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>3 States</td>
<td>Simulation</td>
</tr>
<tr>
<td>Base</td>
<td>50-500</td>
<td>20-50</td>
<td>3 States</td>
<td>Simulation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(2 unstable)</td>
<td></td>
</tr>
<tr>
<td>Lumped (Force Field)</td>
<td>finite</td>
<td>finite</td>
<td>3 States</td>
<td>Heuristic and</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(2 unstable)</td>
<td>Simulation</td>
</tr>
<tr>
<td>Probabilistic Lumped</td>
<td>finite</td>
<td>$\infty$</td>
<td>3 States</td>
<td>Theoretical and</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(2 unstable)</td>
<td>Simulation</td>
</tr>
</tbody>
</table>
we have gathered would be consistent with both modes of activity and only further data with longer runs could distinguish between these two possibilities.

As is generally the case, any research work not only answers some questions, but, in addition, suggests new ones. We now discuss a few open questions that arise. We have shown the usefulness of using deterministic models as simplified versions of probabilistic models. Can this simplification, which we have called an approximate homomorphism, be formalized? In particular, it might be useful to introduce topological ideas into the questions of approximate homomorphisms in order to make this concept precise. That is, the question of how good an approximation we obtain, would be a topic of further study.

In our discussions of the modelling paradigm, we have focussed on two aspects of a four way classification. We have studied a probabilistic base model that has a corresponding simplified lumped model that is either deterministic or probabilistic. What about the case of a deterministic base model? The case of a corresponding deterministic lumped model to represent this type of base model is classical systems theory, but how about a probabilistic lumped model? Ergodic machines could provide an ideal vehicle to study this fourth classification of a deterministic base-probabilistic lumped pairing. Such a modelling situation is, in fact, quite common since the introduction of probabilistic elements often provides a way to examine intractable deterministic base models. This type of situation frequently occurs, for example, in Political Science and Economics.

The study of Ergodic Machines also suggests possible renewed investigation into the concepts of pseudo random number generators. We
have shown that certain properties are sufficient on our ergodic transformations to allow faithful representation of the associated probabilistic automata. Perhaps we can tailor pseudo random number generators for particular problems of interest rather than attempt to find "ideal" pseudo random number generators.

These questions, and others that may suggest themselves to the reader, will hopefully be considered in future research work.

---

APPENDIX I

MATHEMATICAL NOTATION

In this appendix, we describe some of the mathematical notation and notions that we have used in the text. A standard reference for the real analysis is Royden [35]. References for the probability are Breiman [8] and Feller [15].

1. A probability space \((\Omega, \mathcal{F}, P)\) is a set \(\Omega\), a \(\sigma\)-algebra \(\mathcal{F}\) on \(\Omega\), and a probability measure \(P\) on \(\mathcal{F}\).

2. An algebra \(\mathcal{F}\) is a set of sets that is closed under union, intersection and complementation. If the algebra is closed under countable unions also, it is said to be a \(\sigma\)-algebra.

3. Let \(\mathcal{F}\) be a \(\sigma\)-algebra on \(\Omega\). Then \(P\) is a probability measure if \(P(\Omega) = 1\) and \(P\) is a measure. For the concept of a measure, see Royden [35].

4. Given a probability space \((\Omega, \mathcal{F}, P)\), a statement holds almost everywhere (a.e.), if it holds for all \(\omega \in \Omega\), except possibly for a set \(A\) of measure 0, i.e. \(P[A] = 0\).

5. Let \(f: X \to Y\) and \(g: Y \to Z\). Then \(g \circ f\) is function composition. That is, \(g \circ f: X \to Z\) such that \((g \circ f)(x) = g[f(x)]\). \(\text{id}: X \to X\) is the mapping such that \(\text{id}(x) = x\) for all \(x\).

6. \([a,b]\) is the closed interval that includes the two endpoints \(a\) and \(b\). \([a,b]\) excludes the point \(b\) and \((a,b]\) excludes the point \(a\).

7. A set \(A \subset B\) means that \(x \in A \Rightarrow x \in B\). Notice that \(A \subset A\) is a true statement.
8. A function $f$ is monotonically decreasing if for $y > x$,
   
   $f(y) \leq f(x)$. It is strictly monotone decreasing if for $y > x$,
   
   $f(y) < f(x)$.

9. \( \overline{\text{lim}} \) is $\lim$ sup. \( \underline{\text{lim}} \) is $\lim$ inf. See Royden [35], page 36.

10. i.i.d. means independent and identically distributed. The concept refers to a set of random variables on a probability space.
APPENDIX II

THE SIMULATION

The simulation was a package of Fortran programs written to run on the Logic of Computers Group's system. The Logic of Computers Group is a research facility within the Department of Computer and Communication Sciences at The University of Michigan. Its computer facilities include an IBM 1800 with disk bulk storage interfaced to a DEC PDP7 with graphic CRT display.

The simulation package was modularized. There were separate initialization routines for both the base and lumped models. A list of the various subroutines and main programs follows.

BASE: Main program for the base model run
BINOM: Subroutine to calculate a binomial distribution
CNNCT: Helps in setting up the connection pattern
DIST: Computes distribution function for NOISE
GPFIL: Used in file manipulation
INEXT: Allows for various types of external input
INIT: Base Model Initialization
INITL: Lumped Model Initialization
LUMPD: Main program for lumped model run (both deterministic and probabilistic depending on flags set)
NOISE: Generates random noise
PRNT: Print routines (also PRNT1, PRNT2)
RSETB: Used to reset base model state during a run

RSETL: Used to reset lumped model state during a run

THOLD: Generates the threshold function

CHI: Uses output from base and lumped model to calculate chi-square statistics.

FSKIP: Used in file manipulations

SIGNF: Computes approximate significance level for chi-square statistics.

Some routines called from these programs were input/output support routines. For further information, contact the author.
REFERENCES


5. ______, "Decomposition Theory for Automata and Biological Systems," JACC, August 1971. Published in Systems Structure IEEE, pub. no. 71C61-CSS.


14. Ewenczyk, D., "Formalization of the Boyd and Epley Three Stage


45. , Theory of Modelling and Simulation, John Wiley and Sons (forthcoming).