University of Nebraska at Omaha
DigitalCommons@UNO

Student Work

5-1-2002

Reduction of Boolean Networks to Scalar Form.

Christopher Farrow

Follow this and additional works at: https://digitalcommons.unomaha.edu/studentwork

Recommended Citation

Farrow, Christopher, "Reduction of Boolean Networks to Scalar Form." (2002). *Student Work*. 3528.
https://digitalcommons.unomaha.edu/studentwork/3528

This Thesis is brought to you for free and open access by DigitalCommons@UNO. It has been accepted for inclusion in Student Work by an authorized administrator of DigitalCommons@UNO. For more information, please contact unodigitalcommons@unomaha.edu.

Reduction of Boolean Networks to Scalar Form

A Thesis

Presented to the

Department of Mathematics

and the

Faculty of the Graduate College

University of Nebraska

In Partial Fulfillment

of the Requirements for the Degree

Master of Arts in Mathematics

University of Nebraska at Omaha

by

Christopher Farrow

May 2002

UMI Number: EP74726

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

UMI EP74726

Published by ProQuest LLC (2015). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.
All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

THESIS ACCEPTANCE

Acceptance for the faculty of the Graduate College,
University of Nebraska, in partial fulfillment of the
Requirements for the degree Master of Arts,
University of Nebraska at Omaha.

Committee

[signature]	4-19-02	Physics.
Name		Department/School
[signature]	4-19-02	Mathematics
Name		Department/School

Chairperson [signature]

Date April 19, 2002

ACKNOWLEDGEMENTS

For the opportunity to work on this project, I sincerely thank my advisor Dr. Jack Heidel and colleague Dr. Jim Rogers. Furthermore, I would like to extend my gratitude to Dr. Heidel for going extra lengths to insure my place in the Mathematical Cellular Biology Research Group. My gratitude also goes out to Dr. John Maloney for volunteering his time and writing the iteration program and Dr. Wai-Ning Mei of my thesis committee.

Thanks to everyone who was stuck in room 243 in Durham Science Center for the duration of the 2001-2002 school year. You all helped in one way or another.

Special thanks goes out to my loving partner, Heidi Connealy. The best part of school is coming home to you.

This thesis dedicated to my dad, John Farrow. You drive me to succeed.

ABSTRACT

Reduction of Boolean Networks to Scalar Form

Christopher Farrow, M. A.

University of Nebraska at Omaha, 2002

Advisor: Jack Heidel, Ph. D.

This thesis explores methods for finding cycles in synchronous Boolean networks. There is a brief survey the functional iteration method for finding cycles in a synchronous Boolean network. This method can, in theory, completely describe a Boolean network, but is inefficient and unfeasible in practice, especially for large networks. A more direct computational approach, referred to as the computational iterative method, is introduced and the results of this method as applied to three node synchronous Boolean networks are presented. This analysis reveals interesting behavior about these simple networks that in some cases can be generalized to networks of any size. The main focus of the paper is the use of scalar equations, self-referring logic equations that describe the behavior of a node, to determine the possible cycle lengths within a synchronous Boolean network. The scalar equation method is introduced and a theorem is presented that proves the existence of scalar equations contingent on the Jacobian condition, a condition required for the solvability of a system of equations. The reduced scalar equation is presented. The reduced scalar equation is a simplified form of the scalar equation

that can be utilized to determine the possible cyclic structure of a network. In addition, two theorems are introduced that define a class of scalar equations that can be easily reduced to scalar form.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

This paper investigates analytic methods for finding cycles in synchronous Boolean networks. Given an initial starting condition, a synchronous Boolean network has one of two possible long term behaviors. The state of the system can eventually become frozen in time, settling to a fixed state (or fixed point) or the trajectory of the system can get trapped in a loop. In either case, there are methods to detect the future behavior of a synchronous Boolean network given any possible initial state.

There are two indirect methods for determining the cyclic structure of synchronous Boolean networks: the functional iteration method and the scalar equation method. Both of these provide advantages over the direct method of step-by-step determination of the system given an initial state. The functional iteration method can be used to find cycles of any length. This is done by solving systems of equations generated by iteration of the network's logic equations. The scalar equation method uses self-referential logic for each node of the network to determine cyclic structure.

The primary purpose of this paper is to explore methods for finding cycles in large networks. The author is engaged in research to determine the cyclic structure of signal transduction pathways in biochemical networks modelled as synchronous Boolean networks. The particular network of interest has approximately 200 nodes;

well over 10^6 states. Direct determination of the network is prohibited by its size. The functional iteration method is well suited for finding cycles and transients in small networks. However, the computational inefficiency of the method inhibits effective implementation on large networks. Even a moderately sized ten node network poses a challenge for the functional iteration method for in it exists the possibility of cycles of length greater than 1000. The scalar equation method is better suited for large systems, but the method is relatively undeveloped and untested. There are few studies of using scalar equations for analyzing Boolean networks [5,6,7]. In addition, the scalar equation method can provide no advantage for analyzing certain networks with "complicated" scalar equations.

This paper presents advances to the scalar equation method for finding cycles in synchronous Boolean networks. The existence of the scalar equation is proven for a class of networks that comply to the Jacobian condition. This condition, necessary for the solvability of a system of equations, is expected to apply to all synchronous Boolean networks, which makes this a powerful result. The reduced scalar equation is introduced. The reduced scalar equation is a simplified form of the scalar equation that reveals the possible cycle lengths in a network. In addition, a direct computational approach for finding cycles is presented with several results for three node networks. From the results, a conservative upper bound is set for the maximum cycle length of networks of arbitrary size. Although this computational method is not suited for large networks, the results gathered on three node networks can in many cases generalize to networks of larger size. These findings shed light on the structure of synchronous Boolean networks in general.

2. METHODS AND REVIEW OF LITERATURE

2.1 Synchronous Boolean Networks

A Boolean network is a set of n nodes, each of which is either in state 1 (on) or state 0 (off) at any given time t. Each node is then updated at time $t+1$ by inputs from any fixed set of k other nodes according to any desired logical rule. Since each of the n states can be sequentially either off or on, there is a set of 2^n different possible states for the entire network.

Since the total number of states is finite and the network changes states sequentially in discrete time steps, the network must necessarily return to a previously occupied state. With n nodes, the largest possible number of time points before re-encountering a previous state is 2^n. This means that all possible trajectories of the network consist of either cycles (loops) of any length from size one (a fixed point) to a maximum of 2^n, or transient states leading to a cycle.

A useful description of a Boolean network consists of:

A) the nodes with interconnecting arrows indicating which nodes affect each other (a directed graph),

B) a logical table indicating the complete logic that controls each node,

C) a diagram showing the interconnections (i.e. cycles and transient states) between all 2^n states.

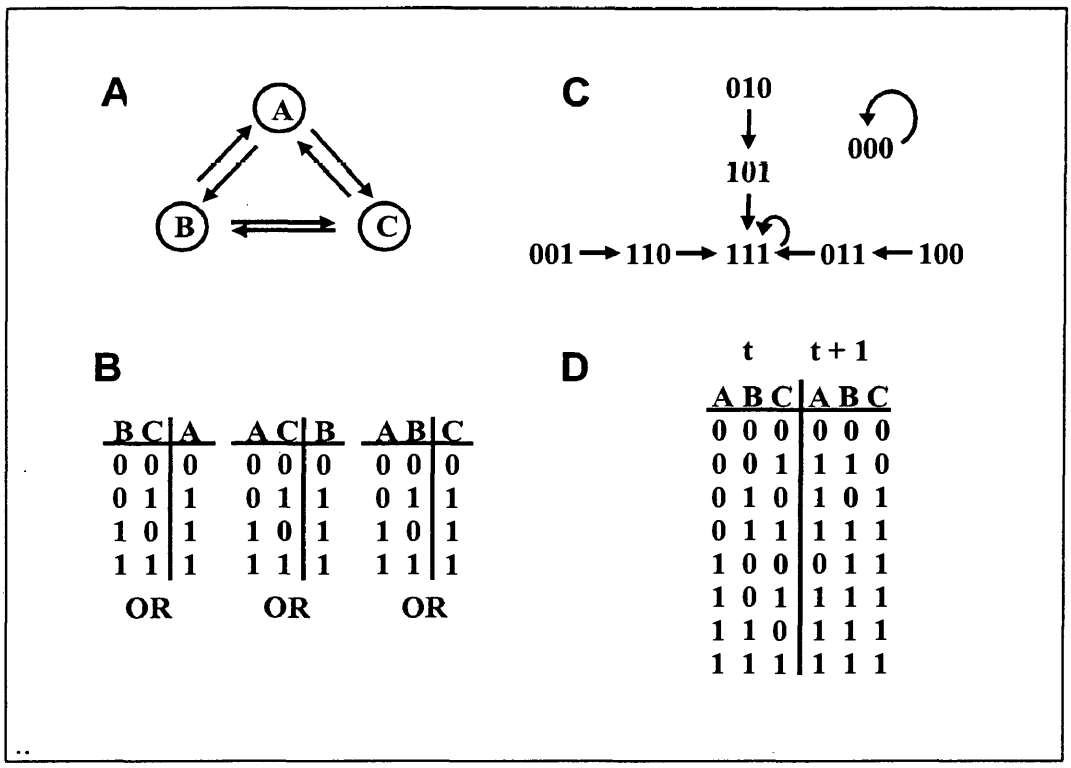

Fig. 2.1: Fully Described Boolean Network

An example of a Boolean network is shown in Figure 2.1. The network shown in Panel A has three nodes, all of which are connected to each other. The logic of the connections is shown in Panel B, all of which are OR. The complete logical description and state space structures ($2^3 = 8$) are easily determined and shown in Panel D, while the map of the trajectories is shown in Panel C.

Of course, this is an ideal total description which can only be realized for small n since one would have to account for 2^n states. Such a complete picture would be unfeasible even for an $n = 10$ network since there would be 1024 possible states. Thus, methods are desired that provide an adequate description of the network utilizing quantities of size n rather than size 2^n without having to completely

enumerate all states. To this end, there are two methods that appear in the literature: the (functional) iteration method and the scalar equation method.

2.2 The Functional Iteration Method

A first step has been taken towards describing synchronous Boolean networks with quantities of size n [2,7]. The logic describing the behavior of each node can be described by a polynomial function in n variables with coefficients of 0 and 1 using Boolean algebra (ordinary arithmetic modulo 2). These n functions, one for each node, encode all of the information about the system. This is most easily understood by looking at some simple examples, such as the one shown in Figure 2.1 on page 4.

In reference to Figure 2.1, the complete system description shown in the diagram can also be obtained with logical function analysis. The logic of each node is OR, so it is easily verified that the logical function for each node is given by:

$$f_A = B + C + BC$$
$$f_B = A + C + AC$$
$$f_C = A + B + AB.$$

Fixed points (cycles of length one) can be determined by setting each logical function equal to the current state as follows:

$$A = f_A = B + C + BC$$
$$B = f_B = A + C + AC$$
$$C = f_C = A + B + AB.$$

This can easily be solved to obtain $A = B = C$ under the rules of Boolean algebra. Thus, this system has two fixed points: $(0, 0, 0)$ and $(1, 1, 1)$.

In a similar manner, antecedents (states that lead to the fixed points) can be determined by setting the logic functions equal to each fixed point:

$$0 = f_A = B + C + BC$$
$$0 = f_B = A + C + AC$$
$$0 = f_C = A + B + AB$$

and

$$1 = f_A = B + C + BC$$
$$1 = f_B = A + C + AC$$
$$1 = f_C = A + B + AB.$$

The first system has only one solution: $(0, 0, 0)$. So, the fixed point $(0, 0, 0)$ has no nontrivial antecedents, as is verified in Panel C of Figure 2.1. The second system of equations yields the points $(0, 1, 1)$, $(1, 0, 1)$, $(1, 1, 0)$, and $(1, 1, 1)$ as the antecedents of $(1, 1, 1)$. The antecedents of the first three antecedents (found by the same method) are $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$, respectively. Thus, working with the three functions f_A, f_B, and f_C (which are determined directly from the three elementary logic tables) the complete state space structure is determined.

Another simple network is shown in Figure 2.2. Unlike the first network, this network contains two periodic orbits and no fixed points. The logic functions for this network are:

$$f_A = 1 + C$$
$$f_B = A$$
$$f_C = B.$$

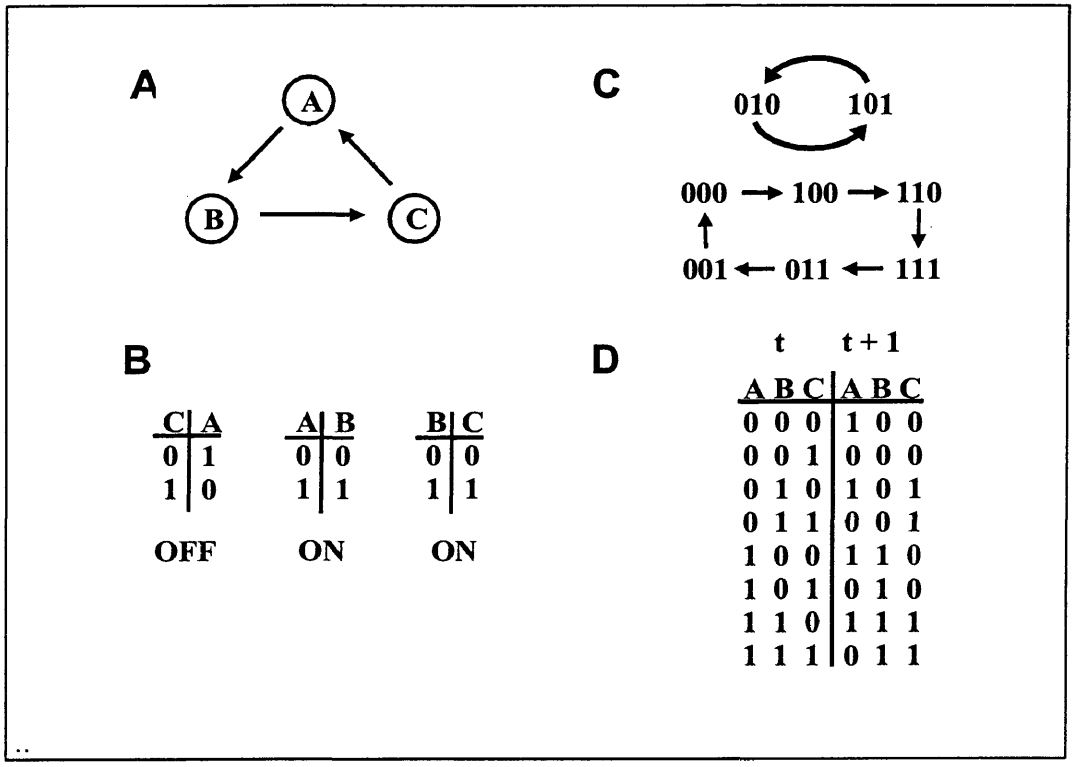

Fig. 2.2: Boolean Network with No Fixed Points

Fixed points are determined by the system:

$$A = f_A = 1 + C$$
$$B = f_B = A$$
$$C = f_C = B.$$

This system is clearly not solvable, so there are no fixed points. Finding periodic points is achieved by composition of the logical functions:

$$A = f_A^2 = 1 + B$$
$$B = f_B^2 = 1 + C$$
$$C = f_C^2 = A.$$

This system has the solutions $(1, 0, 1)$ and $(0, 1, 0)$ which is the period two cycle shown in Panel C of Figure 2.2.

Iteration of the logic functions three and five times (to find period three and period five points, respectively) yields systems that are not solvable, so it can be correctly concluded that there are no states of those types. The fourth iteration generates a system identical to the one solved for period two points. Since period two points are also (trivially) period four points, this result means that there are no true period four points. Similarly, six iterations yields the system of equations:

$$A = f_A^6 = A$$
$$B = f_B^6 = B$$
$$C = f_C^6 = 1 + 1 + C = C.$$

This system is solvable for every one of the eight points in the state space of this network. Since two of the points have been determined to be period two points (which are also trivially period six points), the remaining six points in the state space must comprise a true period six orbit, which is verified to be true in Figure 2.2.

The above algebraic approach using the n node logic functions can be carried out on quite large systems when searching for moderately sized cycles. However, finding all cycles in a large system could involve an enormous number of iterative computations (if very long cycles are present) meaning that the largest cycles of a large system might not be found using this method.

2.3 The Scalar Equation Method

The method of iteration for finding cycles described in the last section is impractical to use for finding all of the cycles in large networks. Often the number of equations necessary to describe the logic of a given system can be reduced to a smaller set of higher order equations or even a single equation, and such a scalar equation is more transparent to analyze for cycles. Specifically, a scalar equation describes the state of a node at a general time t as a function of only that node at earlier times. While the scalar equation technique has been previously applied to Boolean modelling [3], here the scalar form is used to find the number and sizes of cycles in Boolean networks.

Looking again at the network described in Figure 2.2 on page 7, it can be seen that the logic functions that were used previously can be converted to the following time-dependent logic equations:

$$A_{t+1} = 1 + C_t$$
$$B_{t+1} = A_t$$
$$C_{t+1} = B_t.$$

One solves for a scalar equation by progressing the logic equations forward in discreet time steps and substituting out all variables excluding the one representing the desired node. From the above equations, it can be easily verified that $A_{t+3} = 1 + A_t$ (with analogous equations for B_t and C_t). From this simple scalar equation it follows immediately that all elements of the 8-dimensional state space lie on an orbit of period six. It also follows immediately from this equation that there are no period three cycles and no fixed points (which are trivially period three cycles). Thus, the only possibility besides a full period six cycle is a period two cycle. It

has already been shown (and it is easily verified) that a period two cycle exists. Thus, the entire state space consists of a period two cycle and a period six cycle.

The network in Figure 2.2 is a simple case of an affine system (linear terms plus constant terms). Affine Boolean networks have been studied in great detail and their cyclic structure is completely understood in a general way [6,10]. However, since affine systems are a very restricted class of Boolean networks and will only rarely arise in applications, the highly developed theory of affine systems is not used in this paper.

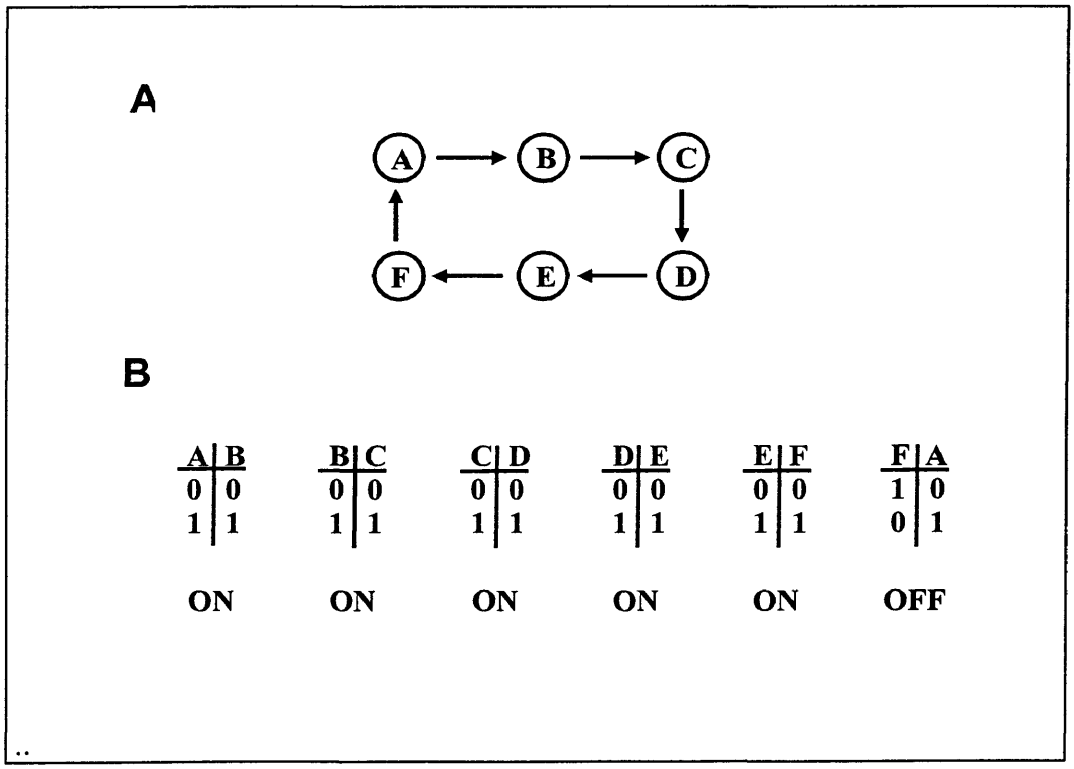

Fig. 2.3: Six Node Affine System

The network shown in Figure 2.3 is another example of an affine system. It is an extension of the previous example as it contains six nodes and analogous logic

(shown in Panel B). The logic for this system can be expressed by the following logic functions:

$$A_{t+1} = 1 + F_t$$
$$B_{t+1} = A_t$$
$$C_{t+1} = B_t$$
$$D_{t+1} = C_t$$
$$E_{t+1} = D_t$$
$$F_{t+1} = E_t.$$

Proceeding as in previous examples, the scalar equation for node A is found to be $A_{t+6} = 1 + A_t$. This means that all states lie on cycles of period 12 and, furthermore, there are no cycles of period six. Thus, there are no cycles of period one (fixed points), two, or three since these would also have period six. Therefore, all cycles have either period four or period 12. Clearly $2^6 = 64 = 5 \cdot 12 + 4$, and the cyclic structure is five distinct cycles of period 12 and one cycle of period four (since the system is affine).

Affine examples such as these can be extended to arbitrarily large size and any single loop whose connections have simple off or on logic can be analyzed in a similar way. Nonlinear networks, however, must be handled differently. There are few general procedures to handle such systems, but it turns out that the scalar approach continues to be very useful.

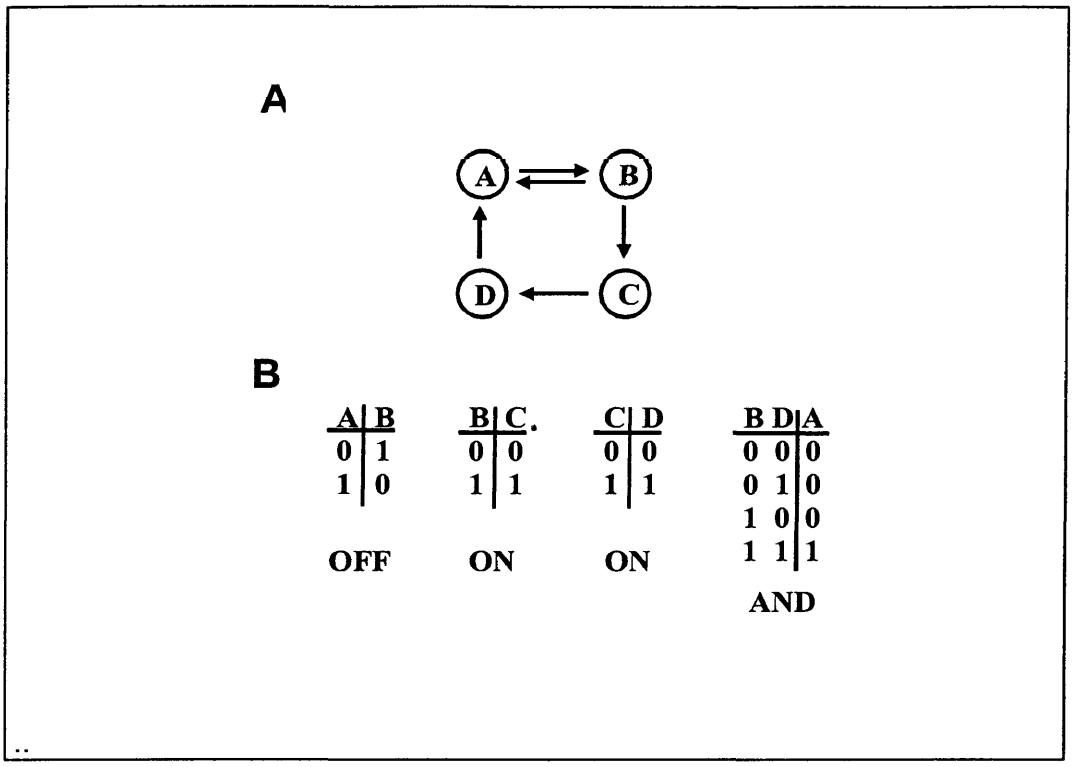

Fig. 2.4: Four Node Nonlinear System

The network shown in Figure 2.4 has the logical equation description:

$$A_{t+1} = B_t D_t$$
$$B_{t+1} = 1 + A_t$$
$$C_{t+1} = B_t$$
$$D_{t+1} = C_t.$$

The scalar equation for node A of this system is $A_{t+4} = (1 + A_{t+2})(1 + A_t)$. This scalar equation separates into two parts where the state of A at $t + 4$ depends on the state of A at both t and $t+2$. The state of A at even time points is dependent only on the previous two even time points, while the state of A at odd time points is dependent on the two previous odd time points. Thus the state of A at the first

two even and odd time points ($t = 0, 1, 2, 3$) must be specified and then the system can move along the trajectory mandated by the logic. For example, consider the even time steps where the values at $t = 0, 2$ are specified and later values are determined from the scalar equation. The possible trajectories at each even time point are given by the following tables:

$t =$	0	2	4
	0	0	1
	0	1	0
	1	0	0
	1	1	0

2	4	6
0	1	0
1	0	0
0	0	1
1	0	0

4	6	8
1	0	0
0	0	1
0	1	0
0	0	1

6	8	10
0	0	1
0	1	0
1	0	0
1	1	0

8	10	12
0	1	0
1	0	0
0	0	1
1	0	0

Regardless of how node A starts, it moves to the period 3 orbit $(0, 0, 1)$ on the even time points. Using the same method, it can be shown that node A moves to the same period three orbit on the odd time points (not shown). When both even and odd times are considered together, there are only two different orbits that are obtained for node A: $(0, 0, 1)$ or $(0, 0, 0, 0, 1, 1)$. That is, the value of node A is in either a period three or period six orbit. Relating B_{t+1}, C_{t+1}, and D_{t+1} to A_t using the logic equations, one obtains either a period three cycle:

$$(0, 1, 0, 1) \rightarrow (1, 1, 1, 0) \rightarrow (0, 0, 1, 1,) \rightarrow (0, 1, 0, 1)$$

or a period six cycle:

$$(0, 1, 0, 0) \rightarrow (0, 1, 1, 0) \rightarrow (0, 1, 1, 1) \rightarrow (1, 1, 1, 1) \rightarrow$$

$$\rightarrow (1, 0, 1, 1) \rightarrow (0, 0, 0, 1) \rightarrow (0, 1, 0, 0).$$

As is typical for nonlinear logics, there are also transient states, in this case seven ($3+6+7 = 16 = 2^4$) transient states, which lead to one or the other of the periodic cycles.

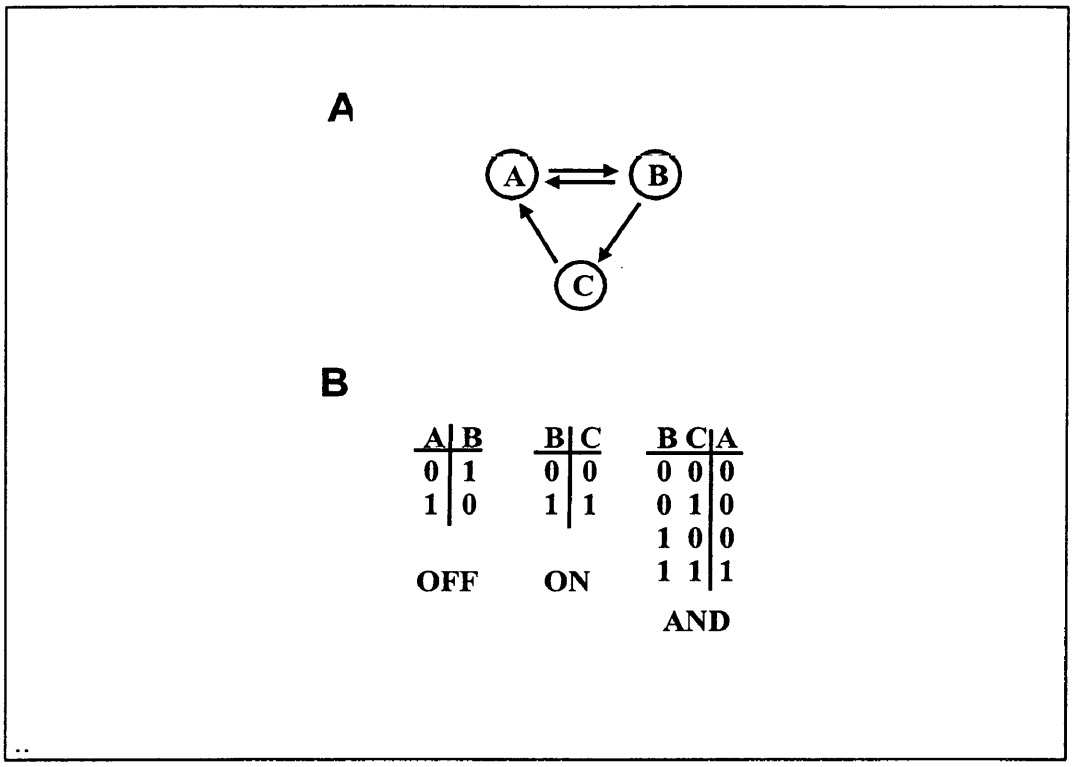

Fig. 2.5: Three Node Nonlinear System

Another example of a nonlinear network is shown in Figure 2.5. The logic functions for this network are:

$$A_{t+1} = B_t C_t$$
$$B_{t+1} = 1 + A_t$$
$$C_{t+1} = B_t.$$

This network would immediately appear to be simpler than the proceeding example and, in fact, it has only one cycle instead of two. Surprisingly, however, one aspect of the analysis is more complicated than for the previous four node system. The scalar equation for node A of this system is $A_{t+3} = (1 + A_{t+1})(1 + A_t)$, with similar equations for B_t and C_t. In order to determine all of the sequential possibilities

for the state of node A in this equation the first three time steps ($t = 0, 1, 2$) must be specified. This is done below:

$t =$								
0	0	0	0	0	1	1	1	1
1	0	0	1	1	0	0	1	1
2	0	1	0	1	0	1	0	1
3	1	1	0	0	0	0	0	0
4	1	1	0	0	1	0	0	0
5	0	0	1	0	1	0	1	0
6	0	0	1	1	0	1	1	1
7	0	0	0	1	0	1	0	1
8	1	1	0	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Notice that node A (eventually) settles into a period five orbit $(0, 0, 0, 1, 1)$ despite the starting condition. Thus, there is the period five cycle:

$$(0,0,0) \rightarrow (0,1,0) \rightarrow (0,1,1) \rightarrow (1,1,1) \rightarrow (1,0,1) \rightarrow (0,0,0)$$

for the full system with three transients ($3 + 5 = 8 = 2^3$).

As demonstrated in the examples above, the scalar equation method can be used to describe the cycles in a synchronous Boolean network. However, as the number of nodes of the system increases, so can the size and complexity of the scalar equations. The analysis used to describe the network in Figure 2.5 would not provide an advantage for analyzing analogous systems of greater size. Observe that the analysis used to determine the cyclic structure of node A required the specification of all (8) possible sequences for the first three time steps of that node. Considering the effort of finding the scalar equations, it would have required less

work to calculate the trajectories of all eight space states directly than it did to find the cycles using the scalar equation method! Just like the functional iteration method, there are cases in which the scalar equation method is infeasible for the analysis of synchronous Boolean networks.

3. RESULTS

3.1 Computational Iteration Method

A computational method has been developed by Dr. John Maloney of the University of Nebraska at Omaha to directly calculate the trajectories of all cycles in a synchronous Boolean network. The implementation of the computational iteration method can be found in the appendix. The code is written in and runs on MAPLE 7. Whereas practical implementation of this program is difficult for large systems, it has been used to analyze the behavior of all three node networks. This analysis brings to light the interesting behavior of these networks. The goal is to understand the intriguing properties found in three node networks and generalize the behavior to networks of any size.

3.1.1 Implementation

This section describes the computational scheme of this iterative method. The program automates the task of describing a Boolean network as described in Section 2.1. The program requires the logical function for each node which describe the interconnectedness and logic of each node. A synchronous Boolean network of size n is defined in a list named *sys*. The nodes are represented sequentially as members of x, a vector of length n. For example, the system described in Fig-

ure 2.1 on page 4 would be represented as $sys := [x[2]+x[3]+x[2]x[3], x[1]+x[3]+x[1]x[3], x[1]+x[2]+x[1]x[2]]$ where $x[1]$, $x[2]$, and $x[3]$ represent nodes A, B, and C, respectively. The list *sys* acts as the interconnected graph (panel A) and the logical tables (panel B) from Figure 2.1. Next, a vector of length 2^n called *nextone* is created and initialized at $\vec{0}$. This vector will hold the space state structure of the network (panel D). The procedure call *iterate*() fills in the vector *nextone* in the following way:

1. The number 0 (k in general) is converted to a binary length n ($\underbrace{00\ldots0}_{n}$). Each bit is stored sequentially in vector x ($[\underbrace{0,0,\ldots,0}_{n}]$). Thus, $x[1] = 0$, $x[2] = 0$, etc.

2. A copy of *sys* is evaluated modulo 2. This gives another vector (length n) of 0s and 1s.

3. The copy of *sys*, treated as a binary number, is converted to decimal.

4. The above decimal number is stored in the 0^{th} position of *nextone*.

5. The process is repeated for the numbers $1, 2, 3, \ldots, 2^n - 1$. The outcome for the k^{th} number is stored in the k^{th} position of *nextone*.

It is important to note that *nextone* stores the states of the system in decimal. For example, the state $(A = 1, B = 0, C = 1)$ is represented in binary as 101 and would be stored in *nextone* as the number 5. If the state $(A = 0, B = 1, C = 1)$ was transient to the state $(A = 1, B = 0, C = 1)$, then the number 5 would be stored in position 3 of *nextone*. In other words, *sys* evaluated with $x = [0, 1, 1]$ (3 in binary) gives $sys = [1, 0, 1]$ (5 in binary).

The map of trajectories can be reproduced by utilizing *nextone* just as one would utilize a table containing the complete logical description of a network. This procedure is automated by the procedures *fixedpoints*() and *cycles*() given in the appendix. The MAPLE call and output for the example in Figure 2.1 is given in Figure 3.1. The fixed points and cycles given by the functional iteration method agree with the results from Section 2.2.

3.1.2 Results on Three Node Networks

Using the above program, the cyclic structure of all three node networks can be determined. When organized correctly, general behavior of these network types is revealed. Important distinctions can be made for the number of nodes with non-linear terms and whether or not any nodes are self-referencing. A self-referencing node has a logistic equation that is dependent upon the earlier state of that node as well as other nodes. Non-linearity is an issue because, given all possible input states, a network with a non-linear node has a reduced number of output states. To see this, consider the hypothetical three node network given by the logic below:

XOR

B	C	A
0	0	0
0	1	1
1	0	1
1	1	0

$B + C = A$

XOR

A	C	B
0	0	0
0	1	1
1	0	1
1	1	0

$A + C = B$

AND

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

$A \cdot B = C$

Node C has non-linear logic **AND** where nodes A and B have linear logic **XOR** (exclusive or). Note has node C has an unbalanced output, i.e. there are unequal

```
> n:=3:  x:='x':
> sys:=[x[2]+x[3]+x[2]*x[3],
> x[1]+x[3]+x[1]*x[3],
> x[1]+x[2]+x[1]*x[2]]:
> x:=vector(3):
> iterate():
> print(nextone);
```

[0, 6, 5, 7, 3, 7, 7, 7]

```
> fixedpoints():
```

The fixed points of nextone are , [0, 7]

```
> cycles():

Report for the value  0

0       0

Report for the value  3

3       7       7

Report for the value  5

5       7       7

Report for the value  6

6       7       7

Report for the value  7

7       7
```

End of cycles

Fig. 3.1: Computational Iterative Implementation

numbers of 0s and 1s. This network has the following complete logical description:

A	B	C	A	B	C
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	0	0	1

As can be seen above, the unbalanced output for node C causes a reduction in the number of output states. The same would hold true for any non-linear logic for C and linear logics for A and B.

Conjecture 3.1 below generalizes the above situation. It can be observed that with all non-linear three node networks without self-referencing, the maximum number of output states given all possible (8) input states is six. This result has been generalized based on the conservative assumption that the non-linear node is a function of only two other nodes. If a non-linear node is a function of more than two other nodes, it is expected that the maximum cycle length is lessened by a larger degree. When used with the functional iteration method, this result eliminates the possibility the largest cycles, those requiring the most work to find.

Conjecture 3.1. An n node synchronous Boolean network with at least one non-linear node and without self referencing cannot have a cycle of length greater than $2^n - 2^{n-2}$.

It would seem that more than one non-linear node would imply a reduced maximum cycle length. This is true to a certain extent, as shown in the next theorem. However, there exists a non self-referencing three node network with two non-linear nodes that has a cycle of length six. This network takes the form:

$$A_{t+1} = B_t + C_t + B_t C_t$$
$$B_{t+1} = 1 + A_t C_t$$
$$C_{t+1} = 1 + A_t + B_t.$$

and has the six-cycle:

$$(0,1,1) \to (1,1,0) \to (1,1,1) \to (1,0,1) \to (1,0,0) \to (0,1,0) \to (0,1,1).$$

One particularly interesting result comes when investigating three node networks with a non-linear term in each node. The result has been verified empirically, but is presented here as a theorem with proof. The structure of the proof does not lend itself to generalization of the result, but it is expected that similar behavior exists in larger non-linear networks. This result, as the one above, puts an upper bound on the possible cycle lengths of certain networks. This is beneficial when finding cycles using the functional iteration method.

Theorem 3.2. A three node synchronous Boolean network with no self-referencing or linear nodes can have no cycle of length larger than five.

Proof. Assume that the system consists of the nodes A, B, and C. Let $a \in \{0,1\}$ be a possible output for node A at time $t+1$. Let $\bar{a}$ be the other element of $\{0,1\}$ ($\bar{a} = (a+1) \mod 2$). Define b, $\bar{b}$, c, and $\bar{c}$ analogously for nodes B and C.

Assume that all possible input states are present at time t. The presence of a non-linear term in the logic of node A means that at time $t+1$ A outputs

either six "a"s and two "$\bar{a}$"s or vice-versa. Without loss of generality, assume A outputs six "a"s and two "$\bar{a}$"s. Since node B has a non-linear term in its logic, it is apparent that B outputs six "b"s and two "$\bar{b}$"s at time $t+1$. In order to get the maximum cycle length, these symbols must be grouped in such a way to get a maximum number of unique groupings. There are three possible arrangements of these symbols:

A	B
$\bar{a}$	b
$\bar{a}$	$\bar{b}$
a	$\bar{b}$
a	b
a	b
a	b
a	b
a	b

A	B
$\bar{a}$	$\bar{b}$
$\bar{a}$	$\bar{b}$
a	b
a	b
a	b
a	b
a	b
a	b

A	B
$\bar{a}$	b
$\bar{a}$	b
a	$\bar{b}$
a	$\bar{b}$
a	b
a	b
a	b
a	b

The first grouping gives the maximum number of combinations (four).

Now consider node C. As with A and B, C gives eight symbols to place; six of them are c and 2 of them are $\bar{c}$. Again, theses symbols must be placed as to get the maximum number of combinations:

A	B	C
$\bar{a}$	b	c
$\bar{a}$	$\bar{b}$	c
a	$\bar{b}$	$\bar{c}$
a	b	$\bar{c}$
a	b	c
a	b	c
a	b	c
a	b	c

It is apparent that this combination gives the maximum number of outputs (five). Since all possible inputs yield a maximum of five outputs, there can be no cycle of length larger than five. □

One example of a non self-referencing entirely nonlinear three node network with cycle of length five is given by the system:

$$A_{t+1} = B_t C_t$$
$$B_{t+1} = C_t + A_t C_t$$
$$C_{t+1} = 1 + A_t + A_t B_t.$$

This networks has the five-cycle:

$$(0,0,1) \to (0,1,1) \to (1,1,1) \to (1,0,1) \to (0,0,0) \to (0,1,1).$$

There is one more result that puts a cap on the maximum cycle length for certain networks. It can be empirically verified that only the three node networks with at least one self-referencing node can have cycles of length eight (the maximum length). Study has begun into the reason for this outcome. It is suspected that

cycles of maximal length in networks of any size are restricted to those networks with at least one self-referencing node.

Conjecture 3.3. An n node synchronous Boolean network with no self-referencing nodes cannot have a maximal cycle (cycle of length 2^n).

The results on three node networks give some insight into the structure of Boolean networks in general. When completely understood, these results could lead to the determination of maximum cycle length in synchronous Boolean networks of any size. This information is helpful when used with the reduced scalar equation of Section 3.2.2, which is used to find the possible cycle lengths within a synchronous Boolean network.

3.2 The Scalar Equation Method

It was shown in Section 2.3 that scalar equations can be used to determine the cyclic structure of synchronous Boolean networks. However, if the scalar equations are complex, as in the network shown in Figure 2.5 on page 14, then determining the actual cycles from the scalar equations provides little or no benefit over the computational iteration method. Techniques are desired that can gather as much information from a scalar equation without resorting to checking all possible sequences of input values. It is shown in Section 3.2.1 that checking all possible sequences of input values often requires 2^n information for a network of size n.

3.2.1 Existence of the Scalar Equation

Given the usefulness of the scalar equation for many networks, it is important to ascertain when scalar equations exist. The main result for this section proves

that a scalar equation exists based on an assumption about the system of logic equations represented at the times $t+1, t+2, \ldots, t+n$ where n is the size of the network. This assumption is the *Jacobian condition* presented in this section. Also important in the theorem is the index of a scalar equation. The *index* of a scalar equation is the degree to which the arbitrary time t has been increased in the scalar equation form. For example, the scalar equation $A_{t+3} = A_t + 1$ has index 3. The proof gives an upper bound on the index of a scalar equation. This indicates the number of time steps for which states must be specified in order to find the cycles via the scalar equation method in Section 2.3.

An example will help to introduce the method of proof. Consider the three node synchronous Boolean network given by the equations:

$$A_{t+1} = B_t + C_t$$
$$B_{t+1} = A_t C_t$$
$$C_{t+1} = A_t + B_t$$

and consider finding the scalar equation for node A. At time $t+3$ there is one equation involving the variable A_{t+3}. That equation is $A_{t+3} = B_{t+2} + C_{t+2}$. This equation involves the undesired variables B_{t+2} and C_{t+2} that cannot appear in the scalar equation for A. In order to eliminate these variables, they must somehow be related back to node A. To this end, consider the two equations with those variables: $B_{t+2} = A_{t+1}C_{t+1}$ and $C_{t+2} = A_{t+1}+B_{t+1}$. These equations introduce the undesired variables B_{t+1} and C_{t+1}. Other equations that involve these variables are $B_{t+1} = A_t C_t$, $C_{t+1} = A_t + B_t$, and $A_{t+2} = B_{t+1} + C_{t+1}$. There is now a system of six equations and six undesired variables. The undesirable variables of this system of equations can be eliminated if the Jacobian with respect to the undesirable variables is non-zero.

Let the system of six equations be given by the vector:

$$\vec{F} = [A_{t+3} + B_{t+2} + C_{t+2}, B_{t+2} + A_{t+1}C_{t+1}, C_{t+2} + A_{t+1} + B_{t+1},$$

$$A_{t+2} + B_{t+1} + C_{t+1}, B_{t+1} + A_tC_t, C_{t+1} + A_t + B_t]$$

and the six undesirable variables be given by the vector:

$$\vec{v} = [B_t, B_{t+1}, B_{t+2}, C_t, C_{t+1}, C_{t+2}].$$

The Jacobian of the system of equations with respect to the undesirable variables is given by the determinant of the matrix:

$$J_{i,j} = \frac{\partial \vec{F}_i}{\partial \vec{v}_j}.$$

The first entry of $J_{i,j}$ given by:

$$J_{1,1} = \frac{\partial \vec{F}_1}{\partial \vec{v}_1} = \frac{\partial(A_{t+3} + B_{t+2} + C_{t+2})}{\partial B_t} = 0.$$

Other values are determined similarly. The complete matrix is given below:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & A_{t+1} & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & A_t & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The determinant is $A_t(1 + A_{t+1})$, which is generally non-zero. Thus, the undesired variables can be eliminated from the system of six equations and a scalar equation for node A can be found. In addition, the index of this equation will be 3 since this is the largest index of all equations utilized.

Theorem 3.4. Assuming the Jacobian condition holds, given an n node synchronous Boolean network, there exists a scalar equation of index n for each node of the network.

Proof. Consider the system at time $t+n$. A scalar equation will be found for node A of the system. The strategy for finding a scalar equation is to eliminate the "undesirable" variables from the equation for A_{t+n}. Undesirable variables are those variables that represent any node other than A.

At time $t+n$ there exists one equation for node A as a function of $n-1$ undesirable variables (other nodes). Now consider the system at time $t+n-1$. This introduces n new equations for n new variables (the state of each node at time $t+n-1$), but only $n-1$ of the new variables are undesirable. It may seem that the equation for A_{t+n-1} is of no use and shouldn't count, but it contains information about the undesirable variables at time $t+n-2$. Collecting all of the equations, there are $n+1$ useful equations relating A with the other variables and $2n-2$ undesirable variables. If the system is considered at time $t+n-2$ there are again n new usable equations and $n-1$ undesirable variables. This can be repeated for the times $t+n-3$, $t+n-4$, and so on.

At time $t+2$ there are $(n-1)^2$ undesirable variables ($n-1$ time steps with $n-1$ undesirable variables per step) and $n(n-2)+1$ usable equations ($(n-2)$ time steps with n equations plus the equation for A_{t+n}). At time $t+1$ there are still $n-1$ new undesirable variables, but now there are only $n-1$ useful equations since the equation for A_{t+1} contains no usable information. Thus, the total tally of undesirable variables is $(n-1)^2+(n-1)=n(n-1)$ and the number of usable equations is $n(n-2)+1+(n-1)=n(n-1)$. Assuming the Jacobian of the system of $n(n-1)$ equations with respect to the $n(n-1)$ undesirable variables

is non-zero (the Jacobian condition), the undesirable variables can be completely eliminated from the system of $n(n-1)$ equations. Thus, an equation for A_{t+n} be found in terms of node A at earlier time steps. □

The Jacobian condition is necessary for the solvability a system of n equations and n unknowns. It is believed that the Jacobian condition holds for all non-trivial Boolean networks, but a proof has yet to be formulated to support this hypothesis. The consequence of such a proof would be that every n node synchronous Boolean network can be reduced to scalar form of index n. Fully describing such a network using the methods of Section 2.3 could require the input of all 2^n initial sequences for each node as in the example corresponding to Figure 2.5. Such cases reduce the usefulness of the scalar equation method, thus a refined scalar equation method is required to handle them.

3.2.2 The Reduced Scalar Equation

Even though the scalar equations for a synchronous Boolean network can be found in certain cases, they may not prove to be useful if they take a complex form. Consider again the example given for the network in Figure 2.2 on page 7. Recall that the scalar equation for node A of that system is given by $A_{t+3} = 1 + A_t$. This equation can be used to show that this network has a cycle of length two and a cycle of length six. Consider advancing this equation forward in time by three time steps. This yields the equation $A_{t+6} = 1 + A_{t+3}$. Substituting this into the original scalar equation (mod 2) gives a new equation $A_{t+6} = A_t$. This more general equation is referred to as the reduced scalar equation. It only has two terms, and the index of the equation gives information about the cyclic structure of the system. The above equation indicates that the all states of node A lie on an

orbit of period six. More specifically, the length of any cycle must be a divisor of 6. So, node A can have a period of 1, 2, 3, or 6. In this example, the information gathered from the reduced scalar equation is less helpful than that of the original scalar equation. Indeed, the reduced scalar equation introduces the possibility of cycles of length three where cycles of this length are prohibited by the scalar equation. However, in networks with scalar equations of higher complexity the reduced scalar equation proves more useful.

Consider the network:

$$A_{t+1} = B_t C_t$$
$$B_{t+1} = 1 + A_t$$
$$C_{t+1} = B_t$$

with scalar equations:

$$A_{t+3} = (1 + A_{t+1})(1 + A_t)$$
$$B_{t+3} = 1 + B_{t+1} B_t$$
$$C_{t+3} = 1 + C_{t+1} C_t.$$

Defining the entire network with these equations would require the enumeration of all initial input sequences (8) for each node. The reduced scalar equations give some information about the possible cycle lengths without this procedure. The reduced scalar equation for each node takes the form $A_{t+5} = A_t$. This implies that this network can have cycles of length 1 or 5. A simple check using the functional iteration method reveals that there are no fixed points, so this network has a cycle of length five and three transients. This simple observation does not define the actual cycles, but it nonetheless gives useful information. Where the number and

size of cycles of a network are of interest, as in the case of the signal transduction model, this analysis provides much gain for little work.

Just as the scalar equations can differ within a network, so can the reduced scalar equations. Consider the network:

$$A_{t+1} = B_t + C_t$$
$$B_{t+1} = A_t$$
$$C_{t+1} = A_t B_t.$$

The scalar equations for this network are:

$$A_{t+3} = A_{t+1} + A_{t+1}A_t$$
$$B_{t+3} = B_{t+1} + B_{t+1}B_t$$
$$C_{t+3} = C_{t+2} + C_{t+1} + C_{t+1}C_t.$$

The reduced scalar equations are:

$$A_{t+4} = A_t$$
$$B_{t+4} = B_t$$
$$C_{t+1} = C_t.$$

These equations imply that this network can have cycles of length 1, 2, or 4.

In the above example not only are two of the three reduced scalar equations distinct, but one of the reduced scalar equations has a lower index than its scalar equation. The method used to prove theorem 3.4 requires increasing the index of the logic equations up to time $t+n$, where n is the size of the network. In order to solve for the reduced scalar equation, one must increment the index of the scalar equation to an index higher than n. This is why the reduced scalar equation is not the same as the scalar equation found from the method of the proof. The

index on each side of the equation is reduced once the reduced scalar equation is found. For example, node A in the above example has a reduced scalar equation of $A_{t+7} = A_{t+3}$ before 3 is subtracted from each index. Since t is arbitrary and only the long-term behavior of the network is of concern (as t approaches 2^n in general), t can be replaced by $t - 3$.

The above form of the reduced scalar equation, $A_{t+7} = A_{t+3}$, may seem superior to the chosen form as it gives some information about the transient states. Namely, this form reveals the number of time steps it takes for transient states to settle into cycles (3) and, therefore, an upper limit on the length of the longest transient trajectory. However, the method used for finding the reduced scalar equation is inexact and can lead to differing equations. For example, if a different method is used to find the reduced scalar equation for the example above, one finds it to be $A_{t+8} = A_{t+4}$. This equation gives different information about the maximum length of a transient trajectory. The information given in this equation is still helpful, but it gives no advantage over the chosen form for finding cycles.

Since the goal is to be able to handle networks of arbitrary size, two general results that predict the possible cycle lengths of synchronous Boolean networks are presented here without proof. Proofs exist for analogous results in number theory [8].

Theorem 3.5. Consider an n node synchronous Boolean network with reduced scalar equations of indices $k_1, k_2, k_3, \ldots, k_n$ and let the least common multiple of these indices be M. Such a network can have cycles of length M, length 1, or of length $m \neq 1$ where m is a divisor of M not relatively prime (sharing no common divisors except 1) to the non-units of $k_1, k_2, k_3, \ldots, k_n$ (excluding 1).

This theorem implies another result more apt for general application.

Corollary 3.6. Given an n node synchronous Boolean network with one known reduced scalar equation of index k, any cycle of that network must have length either dividing k or a multiple of k.

For example, consider a hypothetical three node network with reduced scalar equations:

$$A_{t+3} = A_t$$
$$B_{t+2} = B_t$$
$$C_{t+1} = C_t.$$

This network can only have cycles of length 1 or 6 since 2 and 3 are relatively prime. If only the scalar equation for node A is known, one could deduce that the network can have cycles of length 1, 3, or 6. On the other hand, consider the three node network with reduced scalar equations:

$$A_{t+6} = A_t$$
$$B_{t+6} = B_t$$
$$C_{t+2} = C_t.$$

This network can have cycles of length 1, 2, or 6. Three is not a possible cycle length for the same reason as above. If only the scalar equation for node C is known, possible cycle lengths are 1, 2, 4, 6, and 8.

The above result is of great use when finding cycles with the functional iteration method. In the case that M is prime, it only has two divisors. This implies that the network can only have fixed points and cycles of length M. The former could be easily checked with the functional iteration method. In a general case, the number of positive integer divisors of an arbitrary positive integer M is bounded

from above by $\lceil \frac{M}{2} \rceil + 1$ where $\lceil \rceil$ denotes the next positive integer function. For the above examples, $M = 6$ and the number of divisors of 6 is bounded by $\lceil \frac{6}{2} \rceil + 1 = 4$. Six has four positive integer divisors: 1, 2, 3, and 6.

The results from Section 3.1.2 put an upper bound on the values of $k_1, k_2, k_3, \ldots, k_n$ and, therefore, M. The results of the above theorem uses M to put an upper bound on the number of possible cycle lengths. Thus, these results give a good indication of the types of cycles to expect from a network simply by the form of its logic equations and from its reduced scalar equations.

Given the usefulness of the reduced scalar equation, steps have been made to generalize what forms of the scalar equation can be easily reduced. The following theorems give two classes of affine scalar equation that are easily put into reduced scalar equation form. The following lemma aids in the proof of those theorems.

Lemma 3.7. A scalar equation of the form $A_{t+p} = A_{t+q} + A_t$ where p and q are positive integers and $p > q$ is equivalent to any scalar equation of the form $A_{t+2^m p} = A_{t+2^m q} + A_t$ for non-negative integer m.

Proof. The result obviously holds for $m = 0$. Let $p - q = k$. Adding k to the index of the scalar equation gives $A_{t+p+k} = A_{t+p} + A_{t+k}$. Substituting $A_{t+p} = A_{t+q} + A_t$ yields $A_{t+p+k} = A_{t+q} + A_{t+k} + A_t$. Adding q to the index of this equation gives $A_{t+p+k+q} = A_{t+2q} + A_{t+k+q} + A_{t+q}$ or $A_{t+2p} = A_{t+2q} + A_{t+p} + A_{t+q}$. From the original scalar equation $A_{t+p} + A_{t+q} = A_t$. Thus, $A_{t+2p} = A_{t+2q} + A_t$. Repeating this procedure iteratively gives the result $A_{t+2^m p} = A_{t+2^m q} + A_t$. □

Theorem 3.8. If $p = 2^m q$ with m, p, and q specified as in lemma 3.7, then a scalar equation of the form $A_{t+p} = A_{t+q} + A_t$ can be reduced to the form $A_{t+r} = A_t$ where $r = 2^m p - q$.

Proof. From lemma 3.7, one can write $A_{t+2^m p} = A_{t+2^m q} + A_t$, where m is such that $p = 2^m q$. In other words, $A_{t+2^m p} = A_{t+p} + A_t$. Since $A_{t+p} + A_t = A_{t+q}$, this yields $A_{t+2^m p} = A_{t+q}$. Subtracting q from the indices gives the result $A_{t+2^m p-q} = A_t$. □

For example, consider the scalar equation $A_{t+20} = A_{t+5} + A_t$. Here $p = 20$, $q = 5$, $m = 2$, and $2^m p - q = 75$. Thus, this scalar equation has reduced scalar form $A_{t+75} = A_t$.

Theorem 3.9. If $p = q(2^m + 1)$ with m, p, and q specified as in lemma 3.7, a scalar equation of the form $A_{t+p} = A_{t+q} + A_t$ has the reduced scalar form $A_{t+r} = A_t$ where $r = 2^m p + q$.

Proof. From lemma 3.7, one can write $A_{t+2^m p} = A_{t+2^m q} + A_t$, where m is such that $p = q(2^m + 1)$. Incrementing the indices of this equation by q gives $A_{t+2^m p+q} = A_{t+2^m q+q} + A_{t+q}$. Using $p = q(2^m + 1)$, this becomes $A_{t+2^m p+q} = A_{t+p} + A_{t+q}$. Since $A_{t+p} + A_{t+q} = A_t$, the result is $A_{t+2^m p+q} = A_t$. □

For example, consider the scalar equation $A_{t+27} = A_{t+3} + A_t$. Here $p = 27$, $q = 3$, $m = 3$, and $2^m p + q = 219$. Thus, this scalar equation has the reduced scalar form $A_{t+219} = A_t$.

It is believed that there are several classes of scalar equations with closed form reduced scalar equations. There is currently an effort to find such classes. With a large classification of these forms, one could determine the possible cyclic structure of a synchronous Boolean network by the scalar equations alone. Such an outcome could greatly advance the current techniques for determining the cyclic structure of synchronous Boolean networks and pave the way for new ones.

4. SUMMARY

This paper presents analytic methods for finding cycles in synchronous Boolean networks. The two current methods for determining the cyclic structure of synchronous Boolean networks, the functional iteration method and the scalar equation method, are shown to be insufficient for analyzing general types of networks. The functional iteration method is inadequate for analyzing large networks and the scalar equation method can, in some cases, require as much input information as the direct approach.

Within this paper is proven the existence of the scalar equation for a class of networks. Also, the reduced scalar equation is introduced. The reduced scalar equation is a simplified form of the scalar equation that reveals the possible cycle lengths in a network. The direct computational approach for finding cycles was presented. This method was applied to all three node networks and yielded several results. Some of these results were generalized for large systems including results that place an upper bound on the largest cycle length in a network of arbitrary size. These results are assisting in the development of new methods of analyzing large synchronous Boolean networks.

BIBLIOGRAPHY

[1] Alligood, K. T., Sauer, T. D., & Yorke, J. A. [1997] *Chaos*, Springer, NY.

[2] Cull, P. [1971], "Linear analysis of switching nets", *Kybernetik* **8**(1), 31-39.

[3] Darby, M. & Mysak, L. [1993] "A Boolean delay equation model of an inter-decadal Arctic climate cycle", *Clim. Dyn.* 8, 241-246.

[4] Dunne, P. [1988] *The Complexity of Boolean Networks*, Academic Press, CA.

[5] Heidel, J. & Zhang, F. [1999] "Non-chaotic behavior in three-dimensional quadratic systems II. The conservative case", *Nonlinearity* **12**, 517-633.

[6] Milligan, D. K. & Wilson M. J. D. [1993] "The behavior of affine Boolean sequential networks", *Connect. Sci.* **5**(2), 153-167.

[7] Robert, F. [1986] *Discrete Iterations: A Metric Study*, Springer-Verlag, NY, Chap. 1, pp. 10-11.

[8] Rosen, K. [2000] *Elementary Number Theory*, Addison Wesley Longman, MA, Chap. 9, pp. 307-313, 590.

[9] Tocci, R. J. & Widmer R. S. [2001] *Digital Systems: Principles and Applications*, Prentice Hall, NJ, Eighth edition.

[10] Wilson, M. J. D. & Milligan, D. K. [1992] "Cyclic behavior of autonomous, synchronous Boolean networks: some theorems and conjectures", *Connect. Sci.* **4**(2), 143-154.

[11] Zhang, F. & Heidel, J. [1997] "Non-chaotic behavior in three-dimensional quadratic systems", *Nonlinearity* **10**, 1289-1303.

APPENDIX

```
> convbin:=proc(n::integer,m::posint)
> local u, v, j, k, r, s, t;
> v:=vector(m,0);
> s:=convert(n,binary);
> for j from m to 1 by -1 do
> t:=s mod 10;
> if t=0
> then s:=s/10;
> v[j]:=0;
> else s:=(s-1)/10;
> v[j]:=1;
> fi;
> od;
> r:=eval(v);
> end:
> convnum:=proc(v::vector)
> local u, j, n, m, q, r, s, t;
> m:=vectdim(v);
> u:=vector(m,0);
> q:=1;
> for j from m to 1 by -1 do
> u[j]:=q;
> q:=q*2;
> od:
> s:=0:
> for j from m to 1 by -1 do
> s:=s + u[j]*v[j];
> od:
> t:=s;
> end:
```

```
> iterate:=proc()
> local m1, m2, m3, n, j, k, r, s, t, y;
> global x, sys, nextone;
> m1:=vectdim(sys);
> m2:=vectdim(x);
> m3:=2^m2;
> nextone:=vector(m3,0):
> if m1 <> m2 then print ("ERROR");
> RETURN(); fi;
> y:=vector(m2,0);
> r:=copy(y);
> t:=seq(x[i]=y[i],i=1..m2);
> for j from 1 to m2 do
> r[j]:=subs(t,sys[j]) + 0 mod 2;
> od:
> s:=convnum(r);
> nextone[1]:=s;
> for j from 2 to m3 do
> y:=convbin(j-1,m1);
> t:=seq(x[i]=y[i],i=1..m2);
> for k from 1 to m2 do
> r[k]:=subs(t,sys[k]) + 0 mod 2;
> od:
> s:=convnum(r);
> nextone[j]:=s:
> od:
> end:
```

```
> fixedpoints:=proc()
> local n, m, j, k, r, s, t;
> global nextone;
> t:=[];
> m:=vectdim(nextone);
> for j from 1 to m do
> if nextone[j]=j-1 then if nops(t) = 0 then t:=[j-1];
> else t:=[op(t),j-1]; fi;
> fi;
> od:
> if t<>[] then print (‘The fixed points of nextone are ‘,t);
> else print(‘There are no fixed points‘);
> fi;
> end:
```

```
> cycles:=proc()
> local m, n, n1, j, k, p, q, q1, q2, r, s, t, u, v, Q, S;
> global nextone;
> p:=fopen('c:\\MyDocuments\\cycles.txt',WRITE,TEXT);
> writeline(p,'This is the report on cycles');
> fclose(p);
> m:=vectdim(nextone);
> r:=vector(m,0);
> S:={};  Q:={};
> #start loop 1  ::  loop 1 make an ordered set from nextone
> for j from 1 to m do
> S:=S union {nextone[j]};
> od:
> #end loop 1
> n:=nops(S);
> #start loop 2  ::  loops 2,3,4 check the cycles and print them
> for j from 1 to n do
> t:=S[j];
> q1:=t;
> r[t+1]:=1;
> v:=[t];
> u:=true;
```

```
> #start loop 3  ::  loop 3 makes a list of the cycle
> for k from 1 to m while(u) do
> s:=nextone[t+1];
> v:=[op(v),s];
> if r[s+1]=1 then u:=false;
> else r[s+1]:=1;
> t:=s;
> fi;
> od:
> #end loop 3
> if u then print('There are no cycles for the entry ',v[1]); fi;
> if not u then p:=fopen('c:\\My
> Documents\\cycles.txt', APPEND, TEXT);
> fprintf(p,"%s %g \n","Reportforthevalue", q1);
> printf("\n%s%g\n","Reportforthevalue", q1);
> Q:=Q union {v[nops(v)]}; # Q keeps track of the cycle
> endpoints
> #start loop 4  ::  loop 4 prints out the cycle
> for n1 from 1 to nops(v) do
> q:=v[n1];
> fprintf(p,"%g ",q);
> printf("%g ",q);
> if n1 mod 7 = 0 then fflush(p); fprintf(p," \n"); fi;
> od:
> #end loop 4
```

```
> fprintf(p,"\n");
> fclose(p);
> fi;
> r:=vector(m,0);
> od:
> #end loop 2
> n:=nops(Q);
> v:=vecor(2*n,0);
> #start loop 5  ::  loop 5 creates a zero list, odd entries are
Q
> for j from 1 to n do v[2*j-1]:=Q[j]; od:
> #end loop 5
> #start loop 6  ::  loops 6,7 check the start and stop of each cycle
> for j from 1 to n do
> s:=v[2*j-1]:
> t:=s;
> k:=0:
> u:=true;
> #start loop 7
> for n1 from 1 to m while(u) do
> t:=nextone[t+1];
> if t=s then u:=false;
> if k<>0 then k:=k+1; fi
> else k:=k+1;
> fi;
> od:
> #end loop 7
```

```
> if not u then v[2*j]:=k;  else v[2*j]:=-1000; fi;
> od:
> #end loop 6
> p:=fopen('c:\\MyDocuments\\cycles.txt', APPEND, TEXT);
> fprintf(p,"\n");
> fprintf(p,"\n");
> fprintf(p,"%s  \n","Thecyclesterminateatthepoints");
> #start loop 8
> for n1 from 1 to n do
> q1:=v[2*n1-1];  q2:=v[2*n1];
> fprintf(p,"(%g , %g) ",q1,q2);
> if n1 mod 3 = 0 then fflush(p); fprintf(p," \n"); fi;
> od:
> #end loop 8
> fclose(p);
> print('End of cycles');
> end:
```