Stability of Boolean Dynamical Systems and Graph Periodicity

By:

Víctor A. Ocasio

Thesis submited in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
PURE MATHEMATICS

UNIVERSITY OF PUERTO RICO

MAYAGÜEZ CAMPUS

May 2009

Approved by:

Omar Colón-Reyes, Ph.D President, Graduate Committee	Date
Dorothy Bollmam, Ph.D Member, Graduate Committee	Date
Gabriele Castellini, Ph.D Member, Graduate Committee	Date
Raúl E. Macchiavelli, Ph.D Representative, Graduate Studies	Date
Julio C. Quintana, Ph.D Department Director	Date

Abstract of Thesis to the Graduate School
of the University of Puerto Rico in Partial Fulfillment of the
Requirements for the Degree of Master of Science

Stability of Boolean Dynamical Systems and Graph Periodicity

By

Víctor A. Ocasio González

May 2009

Chair: Omar Colón Reyes

Major Department: Department of Mathematical Sciences

Abstract

In the study of finite dynamical systems it is important to develop efficient algorithms that provide information about the dynamics of the systems. Criteria for determining when a system described by monomials, over the two element field, is a fixed point, have already been determined. We make use of such criteria to study the concept of stability for finite dynamical systems. In order to do this, we use the fact that a monomial dynamical system's cycle structure can be described by the structure of the monomials. This monomial structure can be represented by a digraph. The algorithms presented in this paper, one for stability, the other for fixed points, combine such criteria with the efficiency of depth-first search rendering both algorithms with complexity $O(n^2 log(n))$.

Resumen de Tesis a Escuela Graduada

de la Universidad de Puerto Rico en Requisito Parcial de los

Requerimientos para el grado de Maestría en Ciencias

Stability of Boolean Dynamical Systems and Graph Periodicity

Por

Víctor A. Ocasio González

Mayo 2009

Consejero: Omar Colón Reyes

Departamento: Departamento de Ciencias Matemáticas

Resumen

En el estudio de sistemas dinámicos finitos es importante crear algoritmos que provean información sobre la dinaḿica de los sistemas de manera eficiente. Los criterios para determinar cuando un sistema representado por monomios, sobre el cuerpo de dos elementos, es de punto fijo, ya han sido establecidos. Utilizaremos éstos para estudiar un concepto de estabilidad para sistemas dinámicos finitos. Tomaremos en consideración que la estructura ciclica está completamente definida por su estructura monomial. Esta estructura se representa con un digrafo. Los algoritmos en este escrito, uno para estabilidad y otro para puntos fijos, combinan estos criterios con la eficiencia de búsqueda en profundidad para crear algoritmos con orden $O(n^2 log(n))$.

To my parents, for making me feel as if everything was possible.

Acknowledgements

I would like to thank my advisor, Omar Colón, for helping me develop a healthy sense of pride in my work. Also, my committee for their strict but well-intentioned revisions and advice. I would like to thank my friends for their support and understanding and my love for her unmeasurable patience.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

A finite dynamical system is a function that maps a finite set to itself. In [3], Colón et al., talk about the importance of these systems in genetic modeling and their ability to model the dynamic of gene expressions and relations among genes. This approach enables geneticists to "determine the long term impact of a gene on the other genes," see [6]. Dynamical systems over the field with two elements can be used to model Boolean networks which have applications in both cellular automata and computational biology, see [6]. B. Elspas also mentions in [8] applications of linear dynamical systems in computer control circuits and communications systems. Some of these applications reach a point in time where they do not experience a change in the state they are in. Dynamical systems that model such phenomena are said to reach a *steady state*. It is of great importance to provide methods for efficiently computing when a dynamical system reaches a steady state without having to observe, enumerate or, for all practical purposes, wait until the phenomenon being modeled evolves by itself. A discrete dynamical system that reaches a steady state is called a *fixed point system* ("FPS").

Imagine now we have a set of genes being modeled by a finite dynamical system. Can we *control* the dynamic of the system to transform it to a fixed point dynamic? In other words, is it possible to manipulate the dynamic of a system to achieve a desired

outcome? In the words of Sontag [13], *'to control an object means to influence its behavior so as to achieve a desired goal.'* Therefore, our goal is to determine whether a dynamical system can be controlled in such a way for it to be composed only of steady states. When this can be achieved we say that the dynamical system is *stabilizable*. Mathematically this means that the finite control dynamical system will be endowed with control variables. The quantity of control variables and what they influence will depend on the system. These control variables will influence the behaviour of the dynamical system through a feedback controller. The main interest of this work is to determine if there exists a feedback controller, such that the dynamical system influenced by it reaches a stable state.

In this work we focus on analyzing boolean dynamical systems, since in gene regulatory networks a gene can be viewed as being either active or inactive. The dynamical systems can then be represented as functions map from the n-tuple cartesian product of the two-element field to itself, i.e $f : F_2^n \to F_2^n$. This function can be expressed as a n-tuple of functions and each one of these can be expressed as a polynomial in n variables over F_2, [11]. We focus our study on those f that can be represented by monomials. If we were to consider other types of polynomials the problem of determining whetheher the dynamical system is FPS would become NP-hard [10]. Also, the case where f is a linear dynamical system has already been completely studied and solved, see [12].

In Chapter 2 we establish criteria for determining when a dynamical system is a fixed point system. Using the function f we construct a dependency graph, that is, a digraph, X_f. With this construction we enter the field of computational algebra and use previous results in [1],[2] and [6] to compute an invariant of X_f, called the *loop number*, on its strongly connected components. It has been proven in [5] that if the strongly connected components of X_f all have loop number equal to 1 then f is an FPS. At the end of the chapter we provide a new algorithm of order $O(n^2 log(n))$, where n is the dimension of the system, that determines when a finite dynamical system is a fixed

point system.

We extend these ideas to boolean dynamical control systems in Chapter 3. We define a labeled digraph that will 'label' the vertices in X_f as either *critical* or not. A vertex is critical if it can be influenced by a feedback controller. We use results in [1], [7], [9] and [13] to construct an algorithm of order $O(n^2 log(n))$, that determines when a boolean control dynamical system is stabilizable. In other words, the algorithm determines when we can find a feedback controller such that the dynamical control system can be made to have only steady states.

Remark- The State Spaces in this work were created with the DVD software developed by the Applied Discrete Mathematics Group at the Virginia Bioinformatic Institute (http://dvd.vbi.vt.edu). The Dependency Graphs where created using the Graphviz program developed by AT&T (http://graphviz.org).

Chapter 2

The Fixed Point System Problem

When studying dynamical systems it is only natural to represent the states of the system by a graph in order to see how they interconnect. These connections can be complicated and in examples of dynamical systems in high dimensions, steady state analysis can be messy if not imposible to determine. Is there a better way to represent the dynamic of a system, one that does not involve enumerating all possible transitions? The focus of this chapter is to review the basics of graph theory and relate these ideas to those of a finite dynamical system. At the end of the chapter we propose an algorithm that provides an efficient way to verify if a given finite dynamical system is a fixed point system with complexity $O(n^2 log(n))$.

2.1 Finite Boolean Dynamical Systems and its Graphs

Most of the results presented in this chapter can be found in Colón et. al. [4], [5].

In genetic expression modeling, genes can be viewed as being either active or not. This activity can be measured in a discrete timeline. A particular configuration of the genes in an instant of time can be easily represented by a boolean array. Such an array will have length equal to the number of genes being modeled. For example, if we had

five genes and only the first and third gene were active, we could represent this state of the genes by the array (10100). Any other state can also be represented as a boolean array and all off these arrays are elements of the 5-fold cartesian product of the two-element field with itself. By analogy it can be shown that the state of n genes can then be represented by a boolean array in the n-fold cartesian product of the two element field.

Definition 2.1.1. *A finite boolean dynamical System is a function f from X to itself, $f : X \to X$, where X is the n-fold cartesian product on the two element field with itself, $X = F_2^n$.*

Since there are a finite quantity of possible states in a finite cartesian product of a finite field with itself, using Lagrange's interpolation, it is easy to see that f can be represented as an n-tuple of polynomials in n variables over F_2. The representation will be denoted by $f = (f_1, f_2, \ldots, f_n)$ with every $f_i \in F_2[x_1, x_2, \ldots, x_n]$. Each f_i is a square-free monomial [11]. In other words, for every i, $f_i = \alpha_i x_1^{\epsilon_{1,i}} x_2^{\epsilon_{2,i}} \cdots x_n^{\epsilon_{n,i}}$ where $\epsilon_{j,i} \in \{0, 1\}$ and $\alpha_i \in \{0, 1\}$. Two trivial cases should come to mind. If $\alpha_i = 0$ for every i then $f = 0$. Also, if $\alpha_i = 1$ and $\epsilon_{j,i} = 0$ for every i, j then $f = 1$. Both cases are examples of steady finite boolean monomial dynamical systems and are therefore not be considered in this work. Cases where at least one $\alpha_i = 0$ or $\alpha_i = 1$ with all $\epsilon_{j,i} = 0$ are also not considered for computational symplicity. For the rest of this chapter when talking about *systems* we will mean a finite boolean monomial dynamical system (BMDS).

Returning to the example above, if the configuration of the genes were to change, say the first would turn off and the fifth would activate, then the array changes from (10100) to (00101). Assuming these changes obey a pre-determined set of rules then the state (00101) always comes after state (10100). Therefore, the next state of the system depends on the current state and so forth. Dependence between states can be

illustrated by a directed graph.

Definition 2.1.2. *The composition of f with itself r times, $f^r = \underbrace{f \circ f \circ f \circ \cdots \circ f}_{r-times}$, is denoted as the dynamic of f and is represented by a directed graph, called the state graph (S_f). The graph is constructed such that there is an edge from state a to state b, $a \to b$, if $f(a) = b$.*

Note that by definition $f^r = (f_1^r, f_2^r, \ldots, f_n^r)$ and $f_i^r = \alpha_i (x_1^r)^{\epsilon_{1,i}} \cdots (x_n^r)^{\epsilon_{n,i}}$. A *cycle* is formed when for $r, s \in N$, N the set of natural numbers, $s < r$, $f^r(x) = f^s(x)$ for $x \in F_2^n$. If $r = s + 1$ for some state x then x is a fixed point of f. Recall that this means that the state of the genes stay the same. Since the next state depends on the previous one, it follows that this configuration of the genes stays the same regardless of the passing of time. In other words, gene expressions have steadied. We assume, of course, that the system receives no outside influence. The maximum number m of compositions of f required to enter a cycle is called the *transition* of f with respect to that cycle. This transition number need not be the same for every cycle of f, especially for the non-linear case.

2.1.3 Example: Consider $f(x_1, x_2, x_3) = (x_2, x_1, x_2) : Z_2^3 \to Z_2^3$. Its dynamic is represented by Figure 2.1. Note the 2-cycle between (0,1,0) and (1,0,1).

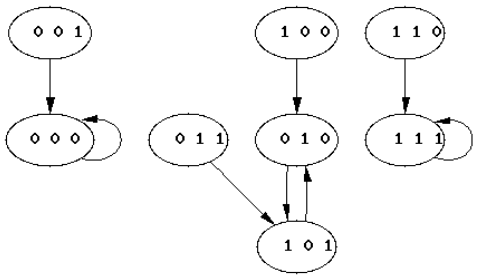


Figure 2.1: A Non Fixed Point System

Since the composition of functions could be performed an infinite amount of times, we now define formally what it means for a system to reach a steady state.

Definition 2.1.4. *A Finite Boolean Dynamical System, (F_2^n, f), is a Fixed Point System if every cycle in its state space is of length 1. Equivalently, if $f^k = f$ for large enough k.*

2.1.5 Example: Consider $f(x_1, x_2, x_3) = (x_2, x_1x_3, x_2x_3) : Z_2^3 \to Z_2^3$. Its dynamic is represented by Figure 2.2.

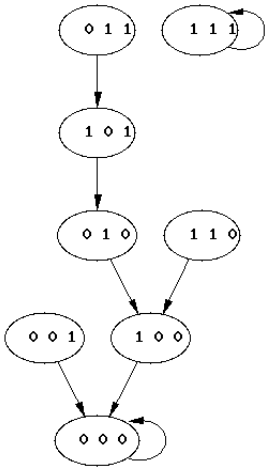


Figure 2.2: A Fixed Point System

In the examples above, $f : Z_2^3 \to Z_2^3$, which means that S_f has 2^3 nodes and edges. In general, a system S_f has 2^n nodes and edges. It can be appreciated how difficult it becomes to analyze the graph when n becomes large. We could miss detecting non-trivial cycles which could lead to the erronous conclusion that a system is an FPS. In fact, it has been stated in [10] that the problem of finding non-trivial cycles for a boolean dynamical system is NP-hard, even if the length of the cycle is 2. This is so basically because this analysis can only be performed after we evaluate the function, f, 2^n times in order to draw the graph. If we wish our mathematics to be useful, in a

computational sense, we need a better way of determining when a function f is an FPS without computing the state space. Colón et al. [2] construct a different graph for the dynamic of f, one that shows the interdependence of the variables instead of the states of the system.

When the function f is composed with itself using the vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ it can be observed how the variables "move" from different f_i. Then, if a cycle forms in this general composition it means that a variable configuration on the f_i repeats itself. It has been proven in [5] that the length of the cycles of the state space of f divides the length of the cycle formed in this general composition of f by itself. The dependency graph of a function f was then created as a means of studying the interdependence of the variables to find the length of the general cycle of the composition of f.

Definition 2.1.6. *Let (F_2^n, f) be a boolean monomial dynamical system. We define the dependency graph of f, X_f, as a digraph composed of a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ and constructed in such a way that there is an edge $v_i \to v_j$ if x_j divides f_i, that is $\epsilon_{j,i} = 1$.*

2.1.7 Example: Consider $f(x_1, x_2, x_3, x_4, x_5, x_6) = (x_2x_3, x_3, x_4, x_1, x_2x_6, x_6)$

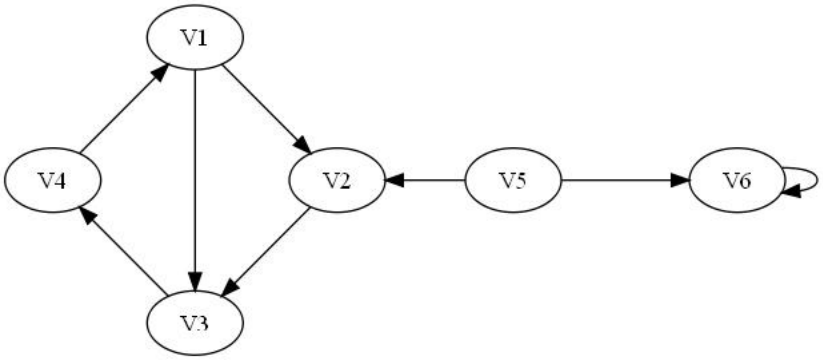


Figure 2.3: Dependency Graph

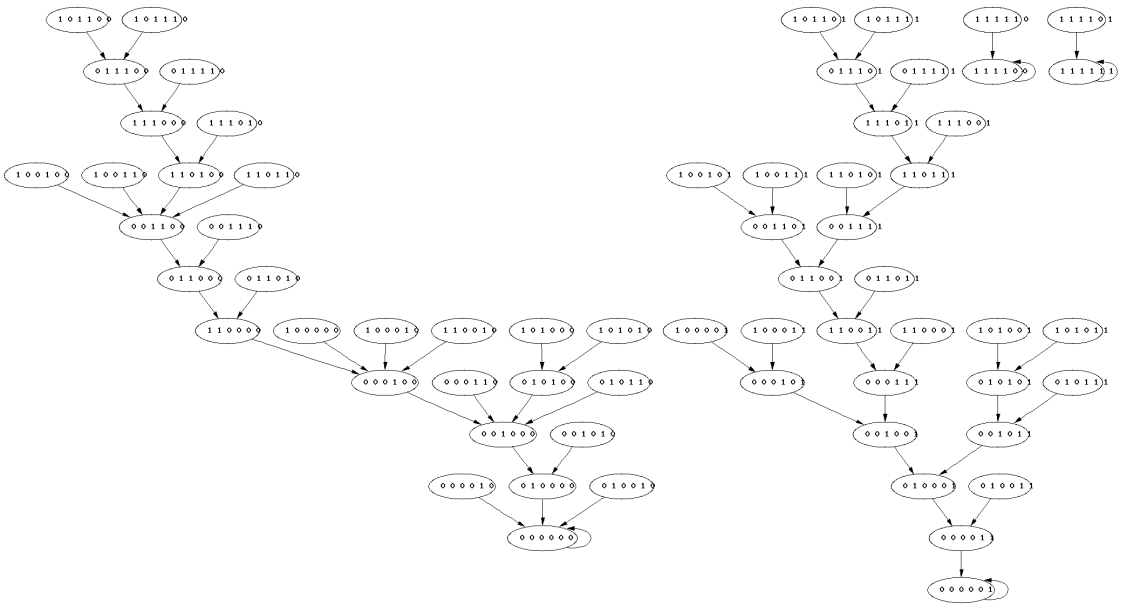


Figure 2.4: State Space

Note that the previous definition permits an edge from a vertex to itself, $a_i \to a_i$, meaning x_i is a factor of f_i. Edges of this sort are called *self-loops*, and they will play an important part in the analysis of the dynamics of the system.

2.2 Strongly Connected Components and the Loop Number

A brief comparison between the state graph and the dependency graph of the example above shows, among other things, that the dependency graph has considerably fewer vertices than the state graph. In fact, X_f has only n vertices while the S_f has 2^n. This is a reduction, computationally speaking, of the problem of determining whether if a given function is a FPS.

The definition of a dependency graph implies that if there exists an edge from one vertex a to another b, then b influences a. Therefore, the edges of the digraph hold information about the interdependency of vertices. Let us then define what it means to move along the digraph.

Definition 2.2.1. *A 'path' p of length r in a graph is a sequence of vertices $(v_1, v_2, \ldots, v_i)$ where every (v_j, v_{j+1}) is connected by an edge. We denote the walk p by $p : v_j \to v_i$ and the length of p by $r = |p|$. If a path begins and ends on the same vertex then it is called a closed path.*

When 'walking' along a path in a digraph it is noticeable that in some digraphs it is impossible to reach some vertices from others. This is an important observation since it defines a relation on the vertices of a graph. Two vertices are connected if there exists a path from one to the other. A set of vertices Y with the property that for every two vertices $v, w \in Y$ there exists a path $p : v \to w$ is called a strongly connected component, SCC, of a graph.

Lemma 2.2.2. *Strongly connected components define an equivalence relation on the set of vertices of a digraph.*

PROOF: We must prove reflexivity, symmetry and transitivity. Let v, w and z be vertices in a strongly connected component.

- Reflexivity: $v \equiv v$ by the empty path of length zero.
- Symmetry: Let $v \equiv w$. By definition of a SCC there exists a path $p : w \to v$, then $w \equiv v$.
- Transitivity: Let $v \equiv w$ and $w \equiv z$ then there exists paths $p : v \to w$ and $q : w \to z$ respectively. Then $qp : v \to z$, thus $v \equiv z$. QED.

2.2.3 Example: Let $f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (x_2, x_3, x_4x_5, x_2x_8, x_6, x_7, x_8, x_5)$. The strongly connected components of X_f are: $\{V_1\}$,$\{V_2, V_3, V_4\}$ and $\{V_5, V_6, V_7, V_8\}$

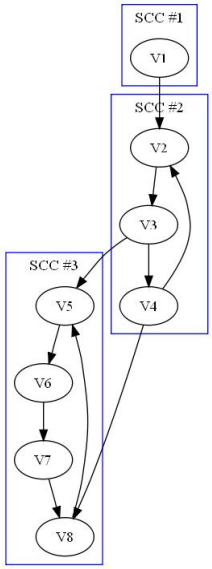


Figure 2.5: X_f divided in SCC

By Lemma 2.2.2, closed paths can exist only inside strongly connected components. We can then analyze the dependency graph one strongly connected component at a time. Earlier, we stated that the length of cycles in the state graph divides the length of the cycle of f evaluated in the vector $\mathbf{x}$. This general cycle is closely related to the length of cycles in a SCC. Also, if after m compositions of the function f with itself we find in a particular f_i an x_j, then there exists a path $p : v_i \to v_j$ of length m, see [4]. As a result, f_i^r is the product of all functions f_j^{r-s} for all walks $p : v_i \to v_j$ of length $r \leq s$. We now define a number for each vertex of the SCC that can be used to establish a closer link between the cycles in a SCC and the state space of the dynamical system it represents.

Definition 2.2.4. *Let X_f be the dependency graph of a finite boolean monomial dynamical system. The loop number, $L(a)$, of a vertex $a \in X_f$ is the minimum of all numbers $t > 0$ such that $t = |p| - |q|$ where $p, q : a \to a$ are closed paths. If no closed path exists from $a \to a$ then the loop number is equal to 0.*

Observe that whenever there exists a closed path $p : a \to a$ of length 1, the loop number of a will also be 1. Take, for example, $p' = pp$, then $|p'| - |p| = 1$. This can be done since the definition of a path does not imply that the vertices in a path cannot be walked repeatedly.

When we calculate the loop number of vertices v_2, v_3 and v_4 in Example 2.2.5 all of them will have the same loop number as v_1. In Example 2.2.6, v_5 and v_6 have the same loop number but are unequal to that of v_1. Note that they belong to different strongly connected components. As shown in Lemma 2.2.7, this is no coincidence.

2.2.5 Example: Consider $f(x_1, x_2, x_3, x_4) = (x_2x_3, x_3, x_4, x_1)$. Then X_f is:

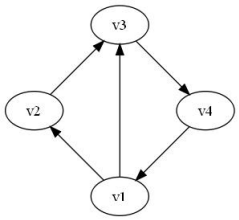


Figure 2.6: Strongly Connected X_f

Let $p : v_1 \to v_2 \to v_3 \to v_4 \to v_1$ and $q : v_1 \to v_3 \to v_4 \to v_1$. Then $L(v_1) = |p| - |q| = 4 - 3 = 1$.

2.2.6 Example: Consider $f(x_1, x_2, x_3, x_4, x_5, x_6) = (x_2x_3, x_3, x_4, x_1x_5, x_6, x_5)$. Then X_f is:

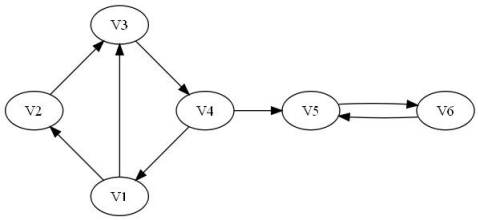


Figure 2.7: X_f with two SCC

$L(v_1) = 1$, $L(v_2) = 1$, $L(v_3) = 1$, $L(v_4) = 1$, $L(v_5) = 2$, $L(v_6) = 2$

For the upcoming results let a strongly connected component be denoted by Y and let $a, b \in Y$.

Lemma 2.2.7. *The Loop Number is invariant on any strongly connected component, Y. Thus, the loop number is a well defined number.*

PROOF: Let $p : a \to a$, $q : a \to a$, $p' : a \to b$ and $q' : b \to a$ be paths such that $|p| - |q| = t$. Then $p'pq', p'qq' : b \to b$ are closed paths with $|p'pq'| - |p'qq'| = t$, so the loop number of b is less than or equal to the loop number of a. By symmetry the loop number is constant on Y. QED.

The next results are the basis of the algorithms to be presented on the next chapter.

Lemma 2.2.8. *Let t be the loop number of Y. Let $p' : a \to b$ and $q' : a \to b$ be paths. Then $|p'| - |q'| \in (t) \subseteq Z$, where (t) is the ideal generated by t in Z, the ring of whole numbers.*

PROOF: Assume $|p'| > |q'|$ and let $|p'| - |q'| = rt + s$ with $0 \leq s < t$. We want to show that $s = 0$. Let $p, q : a \to a$ be such that $|q| - |p| = t$. We have $r \geq 0$. Then, $|p'p| - |q'q| = |p'| + |p| - |p| - |q| = rt + s - t = (r-1)t + s$. Hence there are paths $p'', q'' : a \to b$ with $|p''| - |q''| = s$. Let $p^* : b \to a$ be a path. Then $|p^*p''| - |p^*q''| = s = 0$ because of the minimality of the loop number t. So $|p'| - |q'| \in (t)$.

Corollary 2.2.9. *Let the loop number of Y be t. Let $p : a \to a$ be a closed path. Then, $|p| \in (t)$.*

PROOF: In the previous lemma take $p' = p$ and $q' = pp$.

Careful consideration of Corollary 2.2.9 in conjunction with the definition of a loop number implies that the loop number of Y, $L(Y)$, can be expressed as the greatest common divisor of all closed paths of a given vertex in a strongly connected component. We

can now define an equivalence between the vertices of an SCC, called loop equivalence, such that two vertices $v, w \in Y$, are related if and only if there exists $p : v \to w$ with $|p| \in (t) \subset Z$.

Lemma 2.2.10. *Let v, w vertices of X_f. Define $v \approx w$ if there exists $p : v \to w$ with $|p| \in (t) \subset Z$. Then $\approx$ is an equivalence relation.*

PROOF: We must prove reflexivity, symmetry and transitivity of the relation $\approx$.

1. *Reflexivity:* $a_i \approx a_i$ since the empty path has length zero.

2. *Symmetry:* Let $a_i \approx a_j$ then $\exists p : a_i \to a_j$ with $|p| \in (t)$. Let $q : a_j \to a_i$ be any path. Then $qp : a_i \to a_i$ is a path such that $|qp| = |p| + |p| \in (t)$ by Corollary 2.2.8.

3. *Transitivity:* Let $a_i \approx a_j$ and $a_j \approx a_k$. Then there $\exists p : a_i \to a_j$ and $q : a_j \to a_k$ with $|p|, |q| \in (t)$. Then $qp : a_i \to a_k$ and $|qp| = |p| + |q| \in (t)$. QED.

The number of loop equivalence classes in an SCC Y is equal to the loop number of Y. We now present a collection of results found in [5] that establish the relation between the state space of f and its dependency graph. The idea is to associate the SCC of a dependency graph with the state space of a directed t-gon. The state space of this t-gon is isomorphic to an action on a hypercube in F_2^t. Results that yield as consequence what we present here as Theorem 2.2.13, which is the basis of the algorithm discussed in the next section.

Proposition 2.2.11. The state space of a directed t-gon is isomorphic to the set of orbits of the action of the cyclic group of order t acting on F_2^t, the t-dimensional hypercube, by cyclically exchanging the canonical basis vectors.

Proposition 2.2.12. Let X be strongly connected with loop number $t \geq 1$ and n vertices. Then the subgraph of cycles in the state space of f is isomorphic to the state

space of a directed t-gon and hence to the set of orbits in the hypercube F_2^t under the action of the cyclic group.

Theorem 2.2.13. *Let (F_2^n, f) be a FBMDS. Then f is a fixed point system if and only if the Loop Number of every strongly connected component of X_f is 1.*

The next examples illustrate the results of the theorem. It can be seen in Example 2.2.14 that the dependency graph of f has two strongly connected components, $Y_1 = \{v_1, v_2, v_3\}$ and $Y_2 = \{v_4, v_5\}$. The loop number of each is 1 and therefore the state graph of f has only trivial cycles. In Example 2.2.15 we will remove x_5 from f_5 thus eliminating the edge from v_5 to itself. This will force the loop number of Y_2 to be 2. One can see how 2-cycles form on the new state graph.

2.2.14 Example: Let $f(x_1, x_2, x_3, x_4, x_5) = (x_2, x_1x_3, x_1x_4, x_5, x_4x_5)$

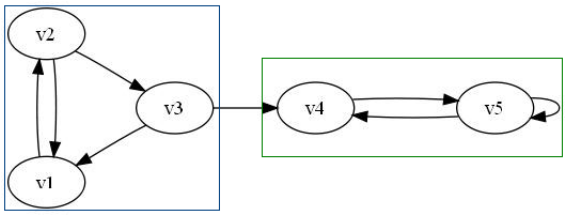


Figure 2.8: Dependency Graph with Loop Number 1 on every SCC

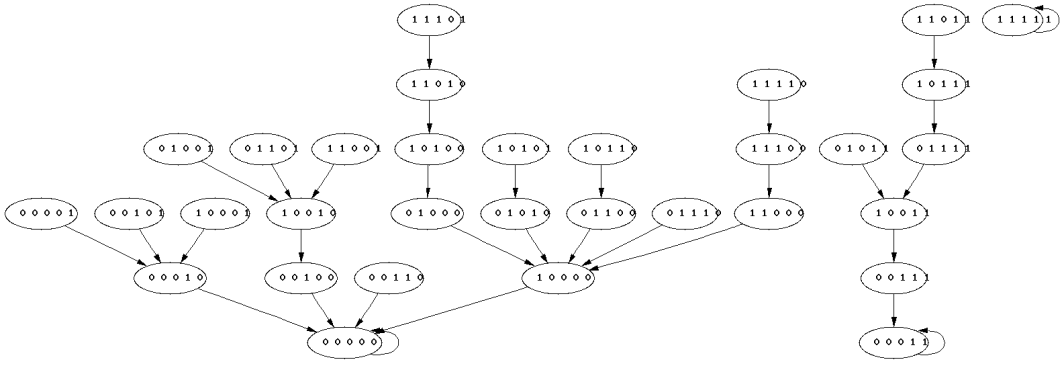


Figure 2.9: State Space of Fixed Point System

2.2.15 Example: Let $f(x_1, x_2, x_3, x_4, x_5) = (x_2, x_1x_3, x_1x_4, x_5, x_4)$

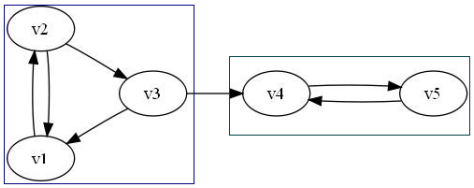


Figure 2.10: Dependency Graph with Loop Number diferent than 1 on a SCC

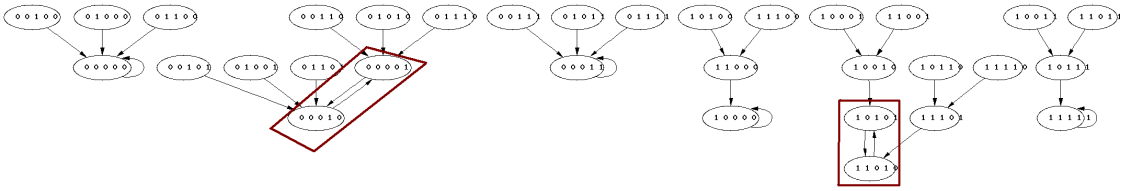


Figure 2.11: State Graph of a Non-FPS

2.3 Algorithm to determine Fixed Point Systems

In Corollary 2.2.9 it was proven that the loop number of a strongly connected component divides the length of every closed path in it. Since the length of all closed paths in an SCC form an ideal in Z, the set of integers, the loop number is the greatest common divisor of all closed paths. Using the fact that the dependency graph can be described by an adjacency matrix of vertices, Coloń et al [5], developed an algorithm to compute the loop number of an SCC. It consists of taking powers of the adjacency matrix and checking for 1's on the main diagonal. Computing a power on the adjacency matrix is equivalent to composing the function f with itself. Therefore, a 1 on the diagonal represents a closed path in the strongly connected component and the power of the matrix the length of the path. The loop number is the greatest common divisor of all the powers of the adjacency matrix, up to the power n, where n the dimension of the dynamical system. Dependency graphs are generally not strongly connected and the algorithm has to be performed once for every SCC yielding complexity $O(n^5 log(n))$.

While effective, the previous algorithm is not efficient in a computational sense, not to mention that one has to decompose the dependency graph manually in strongly connected components before applying the algorithm. So, in order to create a more efficient algorithm two problems have to be overcome, the automatic decomposition of the dependency graph into SCC's and the computation of the loop number for each of them.

Tarjan's depth-first search algorithm is an efficient way to decompose a digraph to strongly connected components. The complexity of depth-first search is $O(n+e)$, where n is the number of vertices and e the number of edges. For a dependency graph the number of edges never exceeds n^2. Therefore, the complexity can be expressed as $O(n^2)$, with n the dimension of the dynamical system. Depth-first search also transforms a digraph into a directed spanning forest, a fact that is useful in computing the loop number.

The calculation of the loop number proved to be a more challenging task. In 1977, E.V. Denardo published [7] where he stated his findings when working with the representation of Markov chains as spanning trees. A *spanning tree* is a digraph with four types of edges, tree edges, back edges, foward edges and cross edges. Tree edges form the spanning tree's general form while back edges, foward edges and cross edges close loops on the digraph (see Example 2.3.1).

Denardo developed an algorithm for computing what he called the *period* of a spanning tree using foward edges, back edges and cross edges. His idea was to separate the spanning tree in levels, as seen in Example 2.3.1, and calculate a number for every edge different from a tree edge. The number of an edge is equal to the length of the loop it closes if the edge is a backward edge, but it is equal to a linear combination of the lenghts of the loops it closes if it is either a cross edge or a foward edge. The number is computed as $level(v) - level(w) + 1$ whenever one encounters a non-tree edge from a node v to a node w. Finally, the period of the graph is be equal to the greatest common

divisor of all those numbers. Clearly, Denardo's period for spanning trees is equivalent to Coloń's et. al. loop number for strongly connected components since walking a digraph using depth-first search produces a spanning forest where every spanning tree is a strongly connected component. The next theorem is proven in [7].

2.3.1 Example: Let $f : F_2^5 \to F_2^5$, $f(x_1, x_2, x_3, x_4, x_5) = (x_2x_3x_5, x_4, x_5, x_1, x_4)$

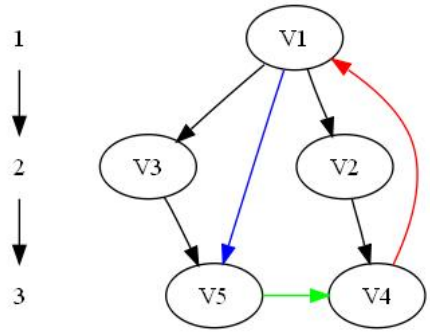


Figure 2.12: X_f as a spanning tree

$v_1 \to v_5$ is a foward edge, $v_4 \to v_1$ is a back edge and $v_5 \to v_4$ is a cross edge.

Theorem 2.3.2. *Let $G = (V, E)$ be a strongly connected graph and T be a spanning tree of G whose nodes are ordered according to a dfs leveling. Then the period of G is $gcd\{level[v] - level[w] + 1 | (v, w) \in E - T\}$.*

The algorithm below is one of the two algorithms presented in this work. It is a modified Tarjan's algorithm that incorporates the insights of Denardo.

input : A directed graph $G = (V, E)$, where E is given by a set of adjacency lists $L[v], v \in V$
output: A list of strongly connected components and their periods

$COUNT \leftarrow 1$
forall $v \in V$ **do**
  Mark v new
  LEVEL[v] $\leftarrow 0$
  P[v] $\leftarrow 0$
  Period $\leftarrow 0$
end
STACK $\leftarrow \oslash$
while *There exists a vertex v marked new* **do**
  SEARCH(v)
end

Algorithm 1: SCC and periods

Proposition 2.3.3. Algorithm 1 calculates the loop number of the scc of a digraph in $O(n^2 log(n))$.

Proof: It is clear that removing every line of code that is involved with calculating the period, the levels or $P(v)$ will transform back the algorithm above into Tarjan's depth-first search algorithm. Our main concern will be to prove that Denardo's theorem hypotesis are met to guarantee that the algorithm calculates the period correctly since the decomposition is done in $O(n^2)$ by depth-first search. First of all, the algorithm must 'level' each node correctly. The leveling is done in Line 8 and it is executed only when a node is visited for the first time, eliminating the possibility of a node having more than one level number assigned. Also, the level number of a node depends on the level number of its parent. This ensures that even if we go all the way to the deepest SCC on a graph, and this is precisely how depth-first search works, each root of a SCC will have the highest level which will not affect the general computation of the $P(v)$, the local period of a node. Since the period of a SCC is an invariant and the gcd is associative, the period of a graph is still the gcd of the local periods of the nodes. Remember that the loop number, which is equivalent to the period, was first defined

```
1  Procedure:SEARCHC(v)

1  Mark v old
2  DFNUMBER[v] ← COUNT
3  COUNT ← COUNT+1
4  LOWLINK[v] ← DFNUMBER[v]
5  Push v on STACK
6  foreach vertex w ∈ L[v] do
7  |  if w is marked new then
8  |  |  LEVEL[w] ← LEVEL[v] + 1
9  |  |  SEARCHC(w)
10 |  |  LOWLINK[v] ← MIN(LOWLINK[v],LOWLINK[w])
11 |  end
12 |  else
13 |  |  if w is on STACK then
14 |  |  |  P[v] ← GCD(P[v],LEVEL[v] + 1 - LEVEL[w])
15 |  |  |  if DFNUMBER[w] < DFNUMBER[v] then
16 |  |  |  |  LOWLINK[v] ← MIN(DFNUMBER[w],LOWLINK[v])
17 |  |  |  end
18 |  |  end
19 |  end
20 end
21 if LOWLINK[v] = DFNUMBER[v] then
22 |  Period ← P[v]
23 |  repeat
24 |  |  Pop x from top of STACK
25 |  |  Period ← GCD(Period,P[x])
26 |  |  Print x
27 |  until x = v
28 |  Print 'End of SCC'
29 |  if Period = 0 then                    /* SCC composed of one vertex */
30 |  |  Period ← 1
31 |  end
32 |  Print Period
33 end
```

Procedure "Modified Tarjan's Search for finding periods"

locally. The gcd of the $P(v)$'s in Line 25 is still necesarry to calculate the correct period since the $P(v)$ calculation on Line 14 assigns the value of $level[v] - level[w] + 1$ to the node on the 'tail' of an edge and non-tree edges not always originate on the same node. But spanning forests have also another type of edge called tree-to-tree edges which are, as there name implies, the ones that connect spanning trees with one another. The algorithm should not compute $P(v)$ for those edges. The conditional on Line 13 garantees that this would not occur since depth-first search uses these edges to go as deep as it can and then 'pops' from the stack any node on a SCC deeper that the one we are in. Also, if a SCC of a graph were unreachable from the others, the leveling would reset itself by going outside of the *SEARCH* procedure, and thus the root would still have the highest level. We have added only conditions and linear commands to Tarjan's depth-first search algorithm and proven that the conditions on Denardo's theorem are met. Therefore the algorithm breaks the graph into SCC and calculates their periods. Calculating the period still requires computing the gcd of two numbers, $gcd(a, b)$. Using Euclid's algorithm requires $O(max(a, b))$ computations and in the case of dependency graphs the longest simple cycle is of lenght n thus the more complex gcd would be between 1 and n. Therefore, Algorithm 1 has complexity $O(n^2 log(n))$. QED.

Chapter 3

Finite Dynamical Control Systems

Sontag said that *'to control an object means to influence its behaviour so as to achieve a desired outcome'* [**?**]. We now know when a finite dynamical system is a fixed point system. We ask ourselves, can any dynamical system be manipulated to become a fixed point system? The answer is no, and in this chapter we develop criteria to determine when such manipulation is possible. Moreover, we will construct an algorithm that, in $O(n^2 log(n))$, will determine if a given finite dynamical system can be stabilized.

3.1 Boolean Dynamical Control Systems

When modeling genetic expression of a disease two important questions come to mind: Does the disease stabilize in time? If not, can we stabilize it? The stabilization of a disease Sdepends on how much control we have over the genes that activate over the course of the disease. With this idea in mind we formally define a boolean control dynamical system.

Definition 3.1.1. *A boolean dynamical control system is a function* $f : F_2^n \times F_2^m \to F_2^n$. *Note that* $f = (f_1, \ldots, f_n)$, $f_i \in F_2[x_1, \ldots, x_n, u_1, \ldots, u_m]$. *We call* $\{x_1, ..., x_n\}$ *the set of state variables and* $\{u_1, ..., u_m\}$ *the set of control variables.*

Observe that since the gene in the i-th position is associated with f_i, then a control variable in f_i means we can somehow "control" the behaviour of the gene. To control a gene means we can influence the interaction that gene has with others. Of course, a gene i influences gene j if x_j divides f_i. Therefore, control variables can be replaced by monomials in $F_2[x_1, \ldots, x_n]$. We define, then, a function $g : F_2^n \to F_2^n \times F_2^m$, called a *feedback controller* such that $g = (u_1, u_2, \ldots, u_m)$ and every $u_i = x_1^{\epsilon_{1,i}} x_2^{\epsilon_{2,i}}, \cdots, x_n^{\epsilon_{n,i}}$ with $\epsilon_{j,i} \in \{0, 1\}$.

3.1.2 Example: Let $f : Z_2^3 \times Z_2^2 \to Z_2^3$, $f(x_1, x_2, x_3, u_1, u_2) = (x_2, x_1x_3u_1, x_2u_1u_2)$.
Let $g : Z_2 \to Z_2^3 \times Z_2^2$, $g(u_1, u_2) = (x_2, x_1)$
Then, $h : Z_2 \to Z_2$, $h(x_1, x_2, x_3) = (x_2, x_1x_2x_3, x_1x_2)$

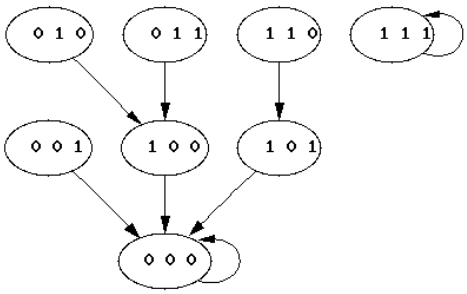
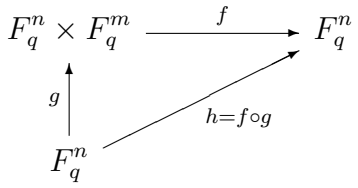


Figure 3.1: X_f of $h = f \circ g$

If we were to graph the state space of the function f without the control variables, that is, seen as a function $f : Z_2^3 \to Z_2^3$, we would soon realize that it is not a fixed point system. Our choice of g for the control variables given by the control system allows us to influence the behaviour of the system in order to stabilize it.

Definition 3.1.3. *Let $f = (f_1, ..., f_n) : F_2^n \times F_2^m \to F_2^n$ be a control system. We say that f is* **stabilizable** *if there exists a function $g : F_2^n \to F_2^n \times F_2^m$, called the* ***feedback controller****, such that $h := f \circ g : F_2^n \to F_2^n$ is a fixed point system. In other words, the next diagram commutes:*



Since $h : F_2^n \to F_2^n$ is a boolean dynamical system we can use the results of the previous chapter to determine if it is a fixed point system. But the stability of f is not guaranteed since g does not always exists. In order to develope criteria for determining when such a g exists we must look to the dependency graphs of both h and f, that is, f seen as $f(x_1, x_2, \ldots, x_n, 1, 1, \ldots, 1) : F_2^n \to F_2^n$.

Definition 3.1.4. *Given two directed graphs X and Y, we say that X is homotopic to Y if there exists a one-to-one correspondence between the vertex set of X and the vertex set of Y and there is an inclusion map from the edge set of X to the edge set of Y.*

Clearly, the dependency graph of f is homotopic to the dependency graph of h. The function h contains all nodes and edges of the dependency graph of f plus all those edges induced by g in the composition. Thus, f is stable if we can add edges to its dependency graph in such a way that every SCC of X_f has loop number 1. We cannot add edges anywhere though. Edges are added only on vertices that correspond to a f_i that is divided by a control variable.

Definition 3.1.5. *Given $f : F_2^n \times F_2^m \to F_2^n$ a boolean control system, consider $\hat{f} := f(x_1, \ldots, x_n, 1, \ldots, 1)$. We define the **labeled dependency graph** of f as the dependency graph of $\hat{f}$, where every vertex v_i corresponding to the function $\hat{f}_i$ is labeled critical if f_i contains a control variable.*

It remains to determine under what arrangement of critical vertices does a labeled dependency graph represent a stabilizable function. Let f and $\hat{f}$ be as in Definition

3.1.5. Assume the dependency graph of $\hat{f}$ to be as in Figure 3.2.

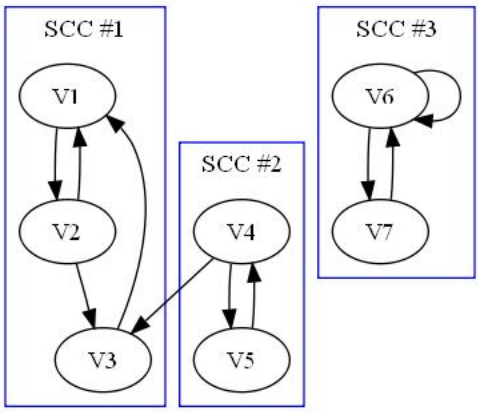


Figure 3.2: Unlabeled X_f

$X_{\hat{f}}$ does not represent the dependency graph of a fixed point system since SCC #2 has loop number equal to two. If $X_{\hat{f}}$ were the dependency graph of a fixed point system, then f would be trivially stabilizable. We cannot decide which vertices are critical but let us assume various configurations of critical vertices. If either V_6 or V_7 were critical vertices, as in Figure 3.3, any edge going from either of them would not change the loop number of SCC #3. This is because the paths used to calculate the minimum number t as in Definition 2.2.3 can still be "walked". Also, any edge to another SCC would be just a tree-to-tree edge and is not taken into account when calculating loop numbers. Therefore, the SCC #2 would still have loop number 2 and we would have to conclude no feedback controller exists such that f is stabilizable.

If some vertex on SCC #2 were critical, say V_4 (see Figure 3.4), then we could construct g such that every control variable would be equal to 1 except one, which would be equal to x_4 thus creating a self-loop in V_4 changing the loop number of SCC #2 to 1. This is not necessarily the only g but we need only one to guarantee stability.

Finally, if a vertex in SCC #1 is critical, say V_1 (see Figure 3.5), then any edge from V_1 to SCC #1 would not change SCC #1 loop number. All edges from V_1 to

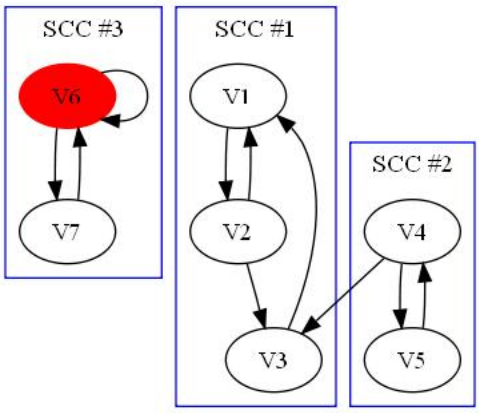


Figure 3.3: Critical vertex on isolated SCC

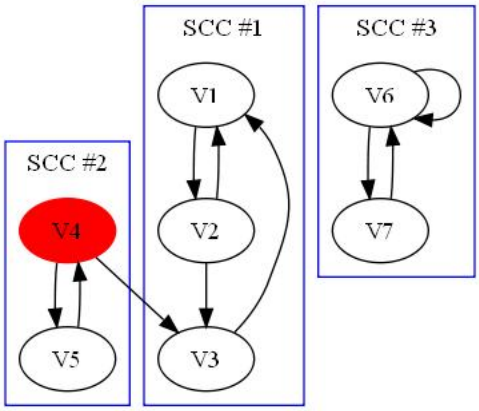


Figure 3.4: Critical vertex on a SCC with $L(SCC) \neq 1$

SCC #3 would be considered tree-to-tree edges and would not be considered for loop number calculation, thus SCC #3 loop number would be the same. On the other hand, an edge from V_1 to SCC #2 would join SCC #1 and SCC #2 into a same SCC. As a consequence, the loop number of V_4 and V_5 would change to 1, stabilizing the dynamical control system.

The next theorem states that the conditions discussed above are not only sufficient but also necessary for a labeled dependency graph to represent a stabilizable boolean

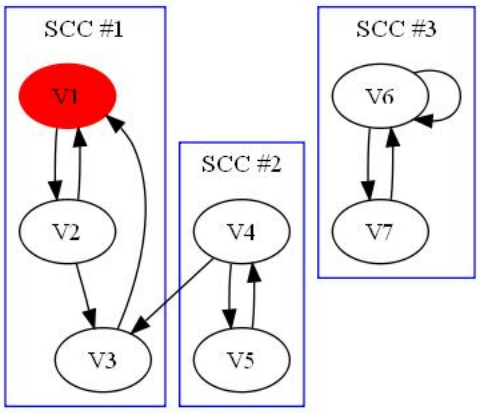


Figure 3.5: Critical vertex on connected SCC

control system.

Theorem 3.1.6. *Let $f : F_2^n \times F_2^m \to F_2^n$ be a boolean monomial control system such that $f(x_1, \ldots, x_n, 1, \ldots, 1) : F_2^n \to F_2^n$ is not a fixed point system. Then f is stabilizable if and only if for every strongly connected component C, of the labeled dependency graph of f with loop number greater than one, either:*

- *C has a critical vertex or*
- *C is connected by a path to a strongly connected component D, which contains a critical vertex.*

PROOF: In order to prove that f is stabilizable we have to prove that we can find a feedback controller $g : F_2^n \to F_2^n \times F_2^m$, such that $h := f \circ g$ is a fixed point system. The construction of a suitable g is straighforward if we look at the labeled dependency graph of f. If C is a strongly connected component of the labeled dependency graph of f with loop number greater than 1, then by hypothesis, it either contains a critical vertex or it is connected by a path to a strongly connected component which contains a critical vertex. In the former case, let v_i be the critical vertex and x_i its corresponding variable. The definition of critical vertex tells us that the function f_i actually depends

on $\vec{u}$, i.e. there is a $j \in \{1, \ldots, m\}$ such that f_i depends on u_j. Now we can add the edge $v_i \to v_i$ to the labeled dependency graph of f. This modification corresponds to setting the function $g_j(\vec{x}) := x_i$. Now we will consider the case where C is connected by a path to a strongly connected component, say D, which contains a critical vertex. Let v_k be the critical vertex contained in D and x_k its corresponding variable. Following the same argument as above, we know there is an $l \in \{1, \ldots, m\}$ such that f_k depends on u_l. Now we can add two edges to the labeled dependency graph of f; namely, the edge $v_k \to v_k$ and an edge that starts at v_k and points to a vertex contained in the component C, say the vertex v_s (which corresponds to the variable x_s). The second edge is not strictly necessary if the component D has loop number equal to 1. Again, this modification corresponds to setting the function $g_l(\vec{x}) := x_k x_s$. We continue this procedure with every strongly connected component of the labeled dependency graph of f that has loop number greater than one. At the end of this process, for some nonempty subset $J \subseteq \{1, \ldots, m\}$ the functions g_t; $t \in J$ will be defined. For the remaining indices in the set $I := \{1, \ldots, m\} \setminus J$ we simply set $g_t \equiv 1$, for all $t \in I$.

Using the feedback controller g obtained, we construct the function h. Clearly, the dependency graph of h differs from the labeled dependency graph of f only in the edges that were added to critical vertices. It is clear that the edges added serve the following purposes: Either they merge two or more strongly connected components into a bigger one or they force the loop number of a strongly connected component to be equal to 1. Therefore, every strongly connected component of h has loop number one and thus h is a fixed point system. This shows that the conditions stated in the theorem are sufficient for the control system F to be stabilized.

To show that the conditions are also necessary, assume that they do not hold. As a consequence, there is a strongly connected component U of the labeled dependency graph of f such that:

(1) U has loop number greater than 1.

(2) U does not contain a critical vertex.

(3) U is not connected by a path to a strongly connected component that contains a critical vertex. Let $v_{i_1}, \ldots, v_{i_t}$ be the set of the vertices contained in U and consider the corresponding variables $x_{i_1}, \ldots, x_{i_t}$ and their update functions $f_{i_1}, \ldots, f_{i_t}$. Since U does not contain any critical vertex, the functions $f_{i_1}, \ldots, f_{i_t}$ cannot depend on any of the control variables $u_1, \ldots, u_m$. Therefore, for *any* feedback controller $g : F_2^n \to F_2^n \times F_2^m$ the function h has the property $h_{i_q} = f_{i_q}, \forall q \in \{1, \ldots, t\}$. In addition, since all arrows starting at the vertices $v_{i_1}, \ldots, v_{i_t}$ must point to vertices in the set $\{v_{i_1}, \ldots, v_{i_t}\}$, none of the functions $f_{i_1}, \ldots, f_{i_t}$ can depend on any of the variables $\{x_1, \ldots, x_n\} \setminus \{x_{i_1}, \ldots, x_{i_t}\}$. Thus, the system h contains the subsystem

$$\bar{h} : F_2^t \to F_2^t$$
$$\vec{\xi} \mapsto \bar{h}(\vec{\xi}) := (f_{i_1}(\vec{\xi}), \ldots, f_{i_t}(\vec{\xi}))$$

As h is iterated, the subsystem $\bar{h}$ is iterated independently from the values of the remaining variables $\{x_1, \ldots, x_n\} \setminus \{x_{i_1}, \ldots, x_{i_t}\}$. The dependency graph of the subsystem consists of a single strongly connected component with loop number greater than 1; and therefore $\bar{h}$ is not a fixed point system. The oscillation of the subsystem $\bar{h}$ is of course observable in the dynamics of h. Summarizing, the system h cannot be a fixed point system. QED

3.2 Algorithm for determining Stability

In the previous section we described the structure of labeled dependency graphs of stabilizable functions. The algorithm discussed in this section determines if a boolean control dynamical system is stabilizable by decomposing its labeled dependency graph. We will again use a modified Tarjan's depth-first search for finding strongly connected

components. The period will still be calculated, using Denardo's results for spanning trees, since Theorem 3.1.6 states that only SCC with loop numbers larger than one must be checked.

input : Labeled connectivity graph $G = (V, E)$ of a Boolean Control Dynamical System where edges are given by adjacency lists
output: A message indicating whether or not the system is stabilizable

```
1  STABILIZABLE ← TRUE
2  COUNT ← 1
3  forall v ∈ V do
4  |  Mark v new
5  |  LEVEL[v] ← 0
6  |  P[v] ← 0
7  |  Period ← 0
8  end
9  STACK ← ⊘
10 while There exists a vertex v marked new do
11 |  SEARCH(v)
12 end
13 if STABILIZABLE then
14 |  Print 'BCDS is STABILIZABLE'
15 end
```

Algorithm 3: Stability for BCDS

```
1  Procedure:SEARCHC(v)

1  Mark v old
2  DFNUMBER[v] ← COUNT
3  COUNT ← COUNT+1
4  LOWLINK[v] ← DFNUMBER[v]
5  Push v on STACK
6  foreach vertex w ∈ L[v] do
7  |  if w marked as critical then
8  |  |  Mark v as critical
9  |  end
10 |  if w is marked new then
11 |  |  LEVEL[w] ← LEVEL[v] + 1
12 |  |  SEARCHC(w)
13 |  |  LOWLINK[v] ← MIN(LOWLINK[v],LOWLINK[w])
14 |  |  if w marked as critical then
15 |  |  |  Mark v as critical
16 |  |  end
17 |  end
18 |  else
19 |  |  if w is on STACK then
20 |  |  |  P[v] ← GCD(P[v],LEVEL[v] + 1 - LEVEL[w])
21 |  |  |  if DFNUMBER[w] < DFNUMBER[v] then
22 |  |  |  |  LOWLINK[v] ← MIN(DFNUMBER[w],LOWLINK[v])
23 |  |  |  end
24 |  |  end
25 |  end
26 end
27 if LOWLINK[v] = DFNUMBER[v] then
28 |  Period ← P[v]
29 |  repeat
30 |  |  Pop x from top of STACK
31 |  |  Period ← GCD(Period,P[x])
32 |  |  Print x
33 |  until x = v
34 |  Print 'End of SCC'
35 |  if Period = 0 then          /* SCC composed of one vertex */
36 |  |  Period ← 1
37 |  end
38 |  if Period > 1 and v is not marked as critical then
39 |  |  STABILIZABLE ← FALSE
40 |  |  Print 'BCDS not Stabilizable'
41 |  |  EXIT
42 |  end
43 end
```


Procedure "Modified Tarjan's Search for determining stability"

Proposition 3.2.1. Algorithm 3 determines the stability of a labeled digraph in $O(n^2 log(n))$ operations.

PROOF: Erasing every line of code that mentions either stability or critical values would transform this algorithm into Algorithm 1. Therefore, this algorithm decomposes the digraph in SCC and calculates their respective periods in $O(n^2 log(n))$ operations, where n the number of vertices. We must prove that the algorithm marks critical vertices and paths to critical vertices correctly. Since the input is a labeled connectivity graph, critical vertices are already marked. The conditionals on Line [7] and Line [14] mark other vertices as critical. These vertices are not critical in the same sense as in Theorem 3.1.6, instead they are marked because they have a path that connects them to a critical vertex. Line [7] verifies if the vertex being visited is critical, in the sense of the algorithm, and marks the vertex on the "tail" of the edge, in this case v, as critical. But it may happen that we are arriving at the vertex w for the first time. If that is so, then v would not be marked critical unless w were critical in the sense of Theorem 3.1.6. In this step we have not analyzed the edges of w. If w were connected to a critical vertex, it would be marked, on an iteration of the SEARCH procedure for w, as critical. That is why we need to verify again on Line [14] for the critical status of w. That way we make sure we verify for critical vertices before and after we have investigated every edge on a vertex. Observe that any vertex of an SCC that has a critical vertex will be marked as critical since it is connected to a critical vertex. In other words, if a root of an SCC is marked as critical then it is critical or is connected to a critical. That is why on Line [38] we can verify in the conditional for the critical status of the root of the SCC that has period bigger than 1 instead of checking every vertex of the SCC. All the markings and verifications of stability are done by conditionals which are of linear order. Therefore, Algoritm 3 determines the stability of a boolean control dynamical system in $O(n^2 log(n))$ operations. QED.

Chapter 4

Discussion of Results

4.1 Conclusions

In this work we have developed criteria to determine when a boolean dynamical control system is stabilizable. We have used previous results on boolean dynamical systems to find necessary and sufficient conditions to determine the stability of these systems. As a result of this, we can now make a complete judgement on the stability of a boolean control dynamical system without having to enumerate all possible state transitions. Theorem 3.1.6 also provides a method for finding an appropiate feedback controller that forces trivial cycles of the state graph of h, in case the control system is stabilizable.

4.1.1 Example: Let $f : Z_2^8 \times Z_2^3 \to Z_2^8$ such that
$f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, u_1, u_2, u_3) = (x_2, x_3x_4, x_1x_5, x_5, x_6u_1u_3, x_5x_7, x_4, x_3x_8u_2)$
Its labeled dependency graph is shown in Figure 4.1.

Theorem 3.1.6 states that $u_2 \equiv 1$ since it can not be used to alter the loop number of any other SCC. Control variables u_1 and u_3 can be made equivalent to x_3 and x_5 respectively. This configuration, while not unique, guarantees the integration of SCC #1 and SCC #2 while at the same time forcing the loop number of the new SCC to be

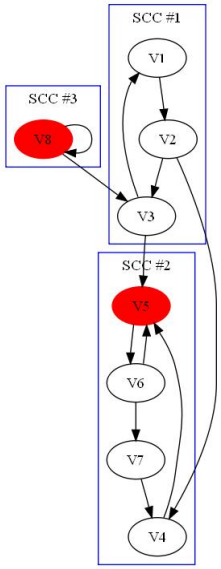


Figure 4.1: Labeled X_f with two critical vertices

1.

Using the results found in [4] and [5] we have developed an algorithm for computing efficiently the loop number of every SCC in a digraph. Algorithm 1 provides a more efficient way of determining when a boolean monomial dynamical system is a fixed point system. The algorithm that was previously known had order $O(n^5 log(n))$ while the algorithm presented here has order $O(n^2 log(n))$. Therefore, Algorithm 1 reduces the computational time by a factor of n^3, a significant improvement. It has been proven in [6] that a finite monomial dynamical system over a finite field with characteristic bigger that 2 requires a booleanization of the system as a method of determining if it is a fixed point system. Algorithm 1 then provides a partial solution for the problem of determining when such systems are fixed point systems.

We have also characterized labeled dependency graphs of stabilizable boolean dynamical control systems. Algorithm 3 determines if such a system is stabilizable. Since the notion of stability is linked to that of fixed point systems Algorithm 3 also provides

a partial solution to the problem of determining when a finite dynamical control system over an arbitrary finite field is stabilizable.

4.2 Future Work

We still need to develop criteria to determine when a finite dynamical control system over an arbitrary finite field is a fixed point system and stabilizable. Work in this area that is being considered is as follows:

- Develop criteria for stability in monomial dynamical systems over finite fields with special characteristics, i.e, $f : F_q^n \times F_q^m \to F_q^n$, where q is a Carmichel prime.
- Develop criteria for stability in monomial dynamical systems over finite fields in general.
- Create algorithm that provides a feedback controller, if it exists, for a boolean control dynamical system.
- Establish necessary and sufficient conditions for determining stability in finite control dynamical system.

Bibliography

[1] Bollman, D.; Colón-Reyes, O.; Ocasio,V.; Orozco,E.; *A Control Theory for Boolean Monomial Dynamical Systems*, Pre-print.

[2] Bollman, D.; Colón-Reyes, O.; Orozco,E.; *Determining When a Monomial Discrete Dynamical System is a Fixed Point System*, Pre-print.

[3] Bollman, D.; Colón-Reyes, O.; Orozco,E.; *Fixed Points in Discrete Models for Regulatory Genetic Networks*, EURASIP Journal of Bioinformatics and Systems Biology, 1 (2007) pp. 8.

[4] Colón-Reyes, O.; *Monomial Dynamical Systems over Finite Fields*, Thesis for Doctoral Degree in Pure Mathematics, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 2005.

[5] Colón-Reyes, O.; Laubenbacher, R.; Pareigis, B.; *Boolean Monomial Dynamical Systems*, Annals of Combinatorics, 8 (2004), pp. 425-439.

[6] Colón-Reyes, O.; Laubenbacher, R.; Jarrah, A.; Strumfelds, B.; *Monomial Dynamical Systems over Finite Fields*, Complex Systems, 16 (2006), pp. 333-342.

[7] Denardo,E.V.; *Periods of Connected Networks and Powers of Nonnegative Matrices*, Math. Oper. Res. 2 (1977), pp.20-24.

[8] Elspas, B.; *The Theory of Autonomous Linear Sequential Networks*, IRE Transactions on the Circuit Theory, CT-6, 5 (1959), pp. 45-60.

[9] Jarvis, J.P.; Shier, D.R.; *Graph-theoretic analysis of finite Markov chains*, In D.R. Shier and K.T. Wallenius , Editors, Applied Mathematical Modeling: A Multidisciplinary Approach, CRC Press(Chapter 13), 1999.

[10] Just, W.; *The steady state system problem is NP-hard even for monotone quadratic Boolean dynamical systems*, Pre-print.

[11] Lidl, R.; Niederreiter, H.; *Finite Fields*, Encyclopedia of Math and its Appl, 20, Cambridge University Press, London, 1977.

[12] Pérez, L.; *Sistemas Dinámicos Booleanos y Operaciones de Puente*, Tesis de Maestría en Matemática Pura, Universidad de Puerto Rico, Mayagüez, 2008.

[13] Sontag, E.; *Mathematical Control Theory*, Texts in applied mathematics 6, 2nd Ed, Springer Verlag, 1998.

[14] Tarjan, R.; *Depth-First Search and Linear Graph Algorithms*, SIAM Journal on Computing 1, 2 (1972), pp. 146-160.