ON THE CONTROL OF ASYNCHRONOUS MACHINES WITH INFINITE CYCLES

By

NIRANJAN VENKATRAMAN

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2004

To my parents, who always stood by me.

ACKNOWLEDGMENTS

First and foremost, I would like to acknowledge the unflagging support and excellent supervision of my advisor, Dr. Jacob Hammer, during my four years at the University of Florida. I thank Dr. Haniph Latchman, Dr. Tan Wong, and Dr. John Schueller for all their efforts and their advice, and for serving on my Ph.D. supervisory committee.

I would like to especially thank Xiaojun Geng (my officemate for three years) for our animated and interesting discussions. I also extend special thanks to my current officemates (Jun Peng and Debraj Chakraborty) for making the last year interesting and full of fun. Special thanks are also due to my roommates (Raj, Nand and Arun) for putting up with my eccentric behavior for four years. I would also like to thank all my friends; their unceasing friendship and care has made this work possible.

Last but not the least, I would like to thank my parents, my grandmother, my sister, her husband and my other relatives for their unfailing love and support during all the years that I have been away from them.

TABLE OF CONTENTS

page

LIST OF TABLES

LIST OF FIGURES

Figure page

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

ON THE CONTROL OF ASYNCHRONOUS MACHINES WITH INFINITE CYCLES

By

Niranjan Venkatraman

August 2004

Chair: Jacob Hammer
Major Department: Electrical and Computer Engineering

My study deals with the problem of eliminating the effects of infinite cycles on asynchronous sequential machines by using state feedback controllers. In addition to eliminating the effects of infinite cycles, the controllers also transform the machine to match the behavior of a prescribed model.

My study addresses machines in which the state is provided as the output of the machine (input/state machines). Algorithms are provided for detecting infinite cycles in a given machine, for verifying the existence of controllers that eliminate the effects of infinite cycles, and for constructing the controllers whenever they exist.

The feedback controllers developed in my study resolve the model matching problem for machines that are afflicted by infinite cycles. They transform the given machine into a machine whose input/output behavior is deterministic and conforms with the behavior of a prescribed model. Thus, the feedback control strategy overcomes the

effects of infinite cycles as well as any other undesirable aspects of the machine's behavior.

Results include necessary and sufficient conditions for the existence of controllers, as well as algorithms for their construction whenever they exist. These conditions are presented in terms of a matrix inequality.

The occurrence of infinite cycles produces undesirable behavior of afflicted asynchronous sequential machines. The occurrence of hazards has been a major problem since the early development of asynchronous sequential machines. There are many methods to design cycle-free machines; but the literature does not seem to contain any technique for eliminating the negative effects of infinite cycles in case it occurs in existing asynchronous sequential machines.

CHAPTER 1
INTRODUCTION

Asynchronous sequential machines are digital logic circuits that function without a governing clock. They are also described variously as clockless logic circuits, self-timing logic circuits, and asynchronous finite state machines (AFSMs). Asynchronous operation has long been an active area of research (e.g., Huffman 1954a, 1954b, 1957), because of its inherent advantages over commonly used synchronous machines. One of the more obvious advantages is the complete elimination of clock skew problems, because of the absence of clocks. This implies that skew in synchronization signals can be tolerated. The various components of the machine can also be constructed in a modular fashion, and reused, as there are no difficulties that are associated with synchronizing clocks. Further, the speed of the circuit is allowed to change dynamically to the maximal respond speed of the components. This leads to improved performance of the machine. Asynchronous machines are by nature adaptive to all variations, and only speed up or slow down as necessary (Cole and Zajicek (1990), Lavagno et al. (1991), Moon et al. (1991), Yu and Subrahmanyam (1992), Fisher and Wu (1993), Furber (1993), Nowick (1993), Nowick and Coates (1994), Hauck (1995)). Asynchronous machines also require lower power, because of the absence of clock buffers and clock drivers that are normally required to reduce clock skew. The switching activity within the machine is uncorrelated, leading to a distributed electromagnetic noise spectrum and a lower average peak value. Moreover, asynchronous operation is inherent in parallel computation systems (Nishimura (1990), Plateau and Atif (1991), Bruschi et al. (1994)).

Asynchronous design is essential to ensure maximum efficiency in parallel computation (Cole and Zajicek (1990), Higham and Schenk (1992), Nishimura (1995)).

Asynchronous machines are driven by changes of the input character, and are self-timed. In this context, it is vital to address design issues, especially those that can lead to potential inconsistencies in the pulses that propagate through the machine. These difficulties are referred to as "hazards" (Kohavi (1970), Unger (1995)). A cycle is one such potential hazard in the operation of asynchronous machines. It causes the machine to "hang," as is demonstrated in software applications with coding defects that can cause the program to hang in an infinite loop. A "critical race" is another hazard that causes the machine to exhibit unpredictable behavior. These hazards can occur because of malfunctions, design flaws, or implementation flaws. The common procedure followed when a hazard occurs is to replace the machine with a new one, built with no defects. In our study, we propose to employ control techniques to devise methods that take a machine out of an infinite cycle and endow it with desirable behavior. Thus, we concentrate on the design of controllers that resolve infinite cycles and control the machine so as to achieve specified performance.

To explain the mode of action of our controllers, a basic description of the structure of an asynchronous machine needs to be reviewed. An asynchronous machine has two types of states: stable states and unstable states. A stable state is one in which the machine may linger indefinitely, while an unstable state is just transitory, and the machine cannot linger at it. A cycle is a situation where the machine moves indefinitely from one unstable state to another, without encountering a stable state. Cycles can occur often in applications, brought about by errors in design, by errors in implementation, or

by malfunctions of components, for example, errors in programming that lead to infinite cycles. The solution currently recommended in the literature for correcting an infinite cycle is to replace the affected machine by a new machine. The solution proposed in our study is, in many cases, more efficient. Moreover, it is the only practical solution in cases where the affected system is out of reach.

The controllers we design are feedback controllers connected to the afflicted machine, as depicted in Figure 1-1. The controller is activated during a cycle. Whenever possible, the controller drives the machine out of the cycle, and leads it to a desirable state. We characterize all cycles from which there is an escape, and present an algorithm for the design of a controller that takes the machine out of any escapable cycle.

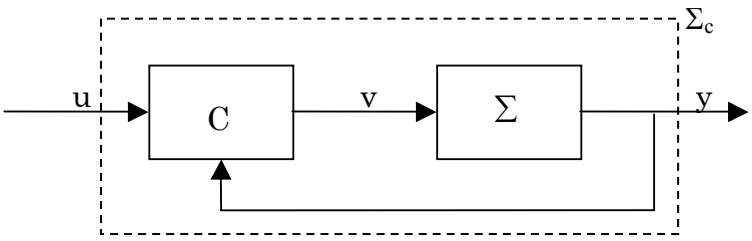

Figure 1-1. Control configuration for the asynchronous sequential machine Σ

Here, Σ is the asynchronous machine being controlled, and C is another asynchronous machine that serves as a controller. We denote by Σ_c the closed loop system represented by Figure 1-1. The controller C eliminates the effects of the cycles of Σ so that the closed loop system Σ_c matches a prescribed model Σ' (i.e., $\Sigma_c = \Sigma'$). The goal is to provide necessary and sufficient conditions for the existence of such a controller C as well as an algorithm for its design. An important property of the controller C is that the closed loop system will function properly whether or not the

cycle actually occurs. As a result, the controller C can be used to prevent the effects of cycles before they occur, improving the reliability of the system.

It is important to note that during a cycle, the machine moves among unstable states in quick succession, without lingering in any of the cycle states. Consequently, it cannot be arranged for an input change to occur at a specific state of the cycle. Thus, the input changes enforced by the controller occur at random during the cycle, and it is not possible to predetermine the state of the cycle at which the controller acts. As a result, the outcome of the controller action is unpredictable in most cases. This could lead to a critical race condition being generated by the controller, which can then be rectified by known procedures (Murphy (1996); Murphy, Geng and Hammer (2002, 2003); Geng 2003; Geng and Hammer (2003, 2004a, 2004b)).

When machine Σ is an input state machine, i.e., the output of Σ is the state of Σ, the controller C is called a state feedback controller. The existence of such a controller C that takes the system out of a cycle depends on certain reachability properties of the system. These reachability properties can be characterized in terms of a numerical matrix called the "skeleton matrix" of the system. The skeleton matrix is calculated from the various transitions of the machine. This skeleton matrix gives a methodology to derive the necessary and sufficient conditions for the existence of a corrective controller C.

The function of the controller is two-fold: it eliminates the effects of the cycle on the machine and assigns to the closed loop a specified behavior. The controller C functions by forcing an input character change while the machine is in a cycle; this may induce a critical race. The outcome of this unpredictable behavior is then used by the controller to assign a proper transition, as required to match the operation of the model.

In this way, the cycles of the machine are stopped, and the machine exhibits the desired external behavior.

The model-matching problem described here is basically a continuation of the work of Hammer (1994, 1995, 1996a, 1996b); Murphy (1996); Murphy, Geng and Hammer (2002, 2003); Geng (2003); and Geng and Hammer (2003, 2004a, 2004b). There are a number of papers that relate to the model-matching problem for synchronous sequential machines (Dibinedetto et al. (1994, 1995a, 1995b, 2001), Barrett and Lafortune (1998)). The studies relating to synchronous machines do not take into consideration the differences between stable and unstable states, fundamental and semi-fundamental mode of operation (Chapter 2), and other issues that are vital to the operation of asynchronous machines. Our study seems to be the first one to explore control theoretic tools to eliminate the effects of infinite cycles on asynchronous machines.

Eliminating the effects of infinite cycles is not the only situation in which model matching is beneficial (Dibenedetto et al. (2001)). For example, most large digital system are designed in a modular fashion, with a number of subsystems. An error in the behavior of the machine detected late in the design phase may make it more economical to use a controller instead of redesigning, or rebuilding, the entire machine. In other words, model matching offers a low-cost option to remedy design and construction errors (Fujita (1993)). Moreover, the controller can also be used to improve the reliability and performance of the system, by deploying it before a fault develops in the system. In this way, the controller can guard against potential malfunctions, while being "transparent" before a malfunction occurs. Thus, in machines where high reliability is vital, our study provides a new design option.

The early work on automata theory was carried out by Turing (1936). Huffman (1954a, 1954b, 1957) investigated various aspects of the synthesis of asynchronous relay circuits. There is a well established technical literature relating to the subject of design of hazard-free machines (e.g., Liu (1963), Tracey (1966), Kohavi (1970), Maki and Tracey (1971)). This is achieved by appropriate state assignment techniques. Classical methods for hazard free state assignments (Huffman (1954a, 1954b)) are reviewed in most textbooks on digital circuit design (e.g., Kohavi (1970)). More recently, studies on the subject were done by Masuyama and Yoshida (1977), Sen (1985), Datta et al. (1988), Nowick and Dill (1991), Fisher and Wu (1993), Chu (1994), Lavagno et al. (1994), Nowick and Coates (1994), and Lin and Devadas (1995). Evading hazards using a locally generated clock pulse was introduced by Unger (1977), Nowick and Dill (1991), and Moore et al. (2000). A graph theoretic approach for state assignment of asynchronous machines was introduced by Datta, Bandopadhyay and Choudhury (1988). Of course, all the above studies discuss the hazard free design of asynchronous machines, and can be applied only before the machine is constructed; only Murphy (1986); Murphy, Geng and Hammer (2002, 2003); Geng 2003; and Geng and Hammer (2003, 2004a, 2004b) detail methods to overcome hazards in an existing asynchronous machine.

Other early works on the synthesis of asynchronous machines were by Mealy (1955) and Moore (1956). These works investigated state and circuit equivalence, and provided methods for synthesizing asynchronous and synchronous sequential machines. Their models of sequential machines are most commonly used today. In the work done by Liu (1963), all internal transitions of a machine go directly from unstable states to terminal states; with no sequencing permitted though stable states. This method of

hazard free assignment (called *single transition time state variable assignment*) uses assignments similar to an equidistant error-correcting code. This kind of assignment, which can be accomplished by mapping the rows of a flow table onto the vertices of a unit n-dimensional cube, is not guaranteed minimal, but it works effectively with incompletely specified flow tables. Tracey (1966) extended this work to techniques that maximize the speed of the machine. A method for sequencing (though unstable states are allowed before a stable state is reached when an input character is applied) was presented by Maki and Tracey (1971), using a technique called the *shared row state assignment method*. This method generally required fewer state variables than single transition time assignments. Other work on hazard-free state assignment techniques can be found in Hazeltine (1965); McCluskey (1965); Armstrong, Friedman and Menon (1968); Hlavicka (1970); Mago (1971); Chiang and Radhakrishnan (1990); Lavagno, Keutzer and Sangiovanni-Vincentelli (1991); Piguet (1991); Oikonomou (1992); Chu, Mani and Leung (1993); and Fisher and Wu (1993).

Some other techniques for avoiding hazards were proposed by Yoeli and Rinon (1964) and Eichelberger (1965). They proposed ternary logic models to analyze certain aspects of the machines. Ternary algebra seems to provide an efficient method for detection and analysis of hazards in sequential circuits. These models were further worked on by Brzozowski and Seger (1987, 1989) and Brzozowski and Yoeli (1987). Another method of avoiding hazards was by generating a clock pulse from other signals in the machine (Bredeson and Hulina (1971), Whitaker and Maki (1992)).

Most applications of asynchronous machines assume that the environment of operation is the fundamental mode (Unger (1969)); that is, an input change is applied

only after the machine has reached stability. No input change is allowed when the machine transits through unstable states. Our study specifies conditions where fundamental mode is not applicable, leading to what is called semi-fundamental mode of operation (Chapter 2). Recent work in asynchronous sequential machines allows the simultaneous change of several input variables. This mode of operation is referred to as *burst-mode* (Davis et al. (1993a), Nowick (1993), Yun (1994), and Oliviera et al. (2000)). Deterministic behavior of the machine in this approach needs special restrictions to be imposed on the machine.

Plenty of literature is available on the application of control theory to automata and asynchronous machines under discrete event systems. One such example that can be readily quoted is supervisory control (Ramadge and Wonham (1987, 1989)). This is based on formal language theory, and assumes that certain events in the system can be enabled or disabled. The control of the system is achieved by choosing control inputs such that the events are enabled or disabled. Literature on discrete event systems is extensive (Ozveren et al. (1991), Lin (1993), Koutsoukos et al. (2000), Alpan and Jafari (2002), Hubbard and Caines (2002), Park and Lim (2002)). Our study employs finite state machines models, which are more suitable to the investigation of engineering implementations than formal language theory (Dibenedetto et al. (2001)).

Asynchronous machines are widely used in industry, as they result in economical digital systems. Some examples of industrial applications are the adaptive routing chip (Davis et al. (1993b)), a cache controller (Nowick et al. (1993)) and the infrared communications chip (Marshall et al. 1994).

The organization of this dissertation is as follows. Terminology and background is provided in Chapter 2. A detailed analysis of infinite cycles, detection algorithms, and the existence conditions are detailed in Chapter 3. This chapter also introduced the concepts of generalized state machines, and the use of transition matrices and skeleton matrices in determining the existence of the controller. Chapter 4 gives a detailed solution to the model matching problem for asynchronous sequential machines with cycles, and an algorithm for the construction of the controller. Chapters 3 and 4 contain the necessary and sufficient conditions for the existence of a controller. An example is solved in Chapter 5, detailing the transition and output functions of the controller states, and finally, a conclusion and possible directions of further research are provided in Chapter 6.

CHAPTER 2
TERMINOLOGY AND BACKGROUND

2.1 Asynchronous Machines and States

Here, the background and notation is introduced and the basic theory of asynchronous sequential machines is described. An asynchronous machine is activated by a change of its input character; it generates a sequence of characters in response. To explain the mathematical theory, we introduce the notation and terminology involved in describing sequence of characters.

An *alphabet* is a finite non-empty set A; the elements of this set are called *characters*. A *word* w over an alphabet A is a finite and ordered (possibly empty) string of characters from A. Let A be a finite non-empty alphabet, let A^* denote the set of all strings of characters of A, including the empty string, and let A^+ be the set of all non-empty strings in A^*. The length $|w|$ of a string $w \in A^*$ is the number of characters of w. For two strings $w_1, w_2 \in A^*$, the concatenation is the string $w := w_1w_2$, obtained by appending w_2 to the end of w_1. A non empty word w_1 in A^* is a *prefix* of the word w_2 if $w_2 = w_1w$; and w_1 is a *suffix* of the word w_2 if $w_2 = w'w_1$, where $w, w' \in A^*$. A *partial function* $f : S_1 \rightarrow S_2$ is a function whose domain is a subset of S_1 (Eilenberg (1974)).

We now introduce the concept of a sequential machine. A sequential machine provides a working mathematical model for a computing device with finite memory. It has the following characteristic properties:

- A finite set of inputs can be applied to the machine in a sequential order.
- There is a finite set of internal configuration that the machine can be in. These configuration are called *states*, and physically, these states correspond to the setting on the flip flop or the bit combinations in memory devices.
- The current internal configuration or state and the applied input character determine the next internal configuration or state that the machine achieves. For deterministic systems, the next state is unique.
- There is a finite set of input characters.

There are two broad classes of sequential machines: asynchronous machines and synchronous machines. An *asynchronous sequential machine* responds to input changes instantaneously as they occur. On the other hand, a *synchronous sequential machine* is driven by a clock, and the values of input variables affect the machine only at clock "ticks". The current work concentrates on asynchronous sequential machines. These machine describe the structure of the fastest computing systems.

An *asynchronous machine* is defined by the sextuple $\Sigma := (A,Y,X,x_0,f,h)$, where A, Y, and X are nonempty sets, x_0 is the initial state, and $f : X\times A \rightarrow X$ and $h : X\times A \rightarrow Y$ are partial functions. The set A is the set of *input values,* Y is the set of *output values*, and X is the set of *states.* The partial function f is the *recursion function* and h is the *output function*. The operation of the machine is described by the recursion

$$x_{k+1} = f(x_k, u_k),$$
$$y_k = h(x_k, u_k), \quad k = 0, 1, 2,\ldots \tag{2-1}$$

A *valid pair* $(x,u) \in X\times A$ is a point at which the partial functions f and h are defined. We assume that the initial state x_0 is provided with Σ. An input sequence is *permissible* when all pairs (x_k, u_k), $k = 0, 1, 2, \ldots$ are valid pairs. The integer k acts as the step counter. An increment in the step counter takes place with every change of

the input value or with a state transition. For consistency, we assume that the functions f and h have the same domain. It is clear from the above description that the recursion (2-1) represents a causal system. When the output function h only depends on the state and does not depend on the input, the machine induces a strictly causal system.

If the functions f and h are functions rather than partial functions, the machine is said to be complete. A deterministic finite state machine is often referred to as the *Mealy machine* (Mealy (1955)). If the output function does not depend on the input variable, then the machine is referred to as the *Moore machine* (Moore (1956)). An asynchronous Mealy machine can be transformed into an equivalent Moore machine, and vice versa.

A *non-deterministic finite state machine* $\Sigma = (A,Y,X,x_0,f,h)$ has the following objects:

- A finite, non-empty set A of permissible input characters, termed as the *input alphabet*;
- A finite, non-empty set Y of permissible output characters, termed as the *output alphabet*;
- A finite, non-empty set X of states of the machine. Any number of the states can be designates as initial or terminal states;
- A non-deterministic partial *transition function* $f : \beta(X)\times A \rightarrow \beta(X)$, where $\beta(X)$ denotes the set of all the subsets of X (i.e., the power set of X);
- A partial function $h : X\times A \rightarrow Y$, called the *output function*; and
- The initial state of the machine $x_0 \in X$.

The transition function f can generate sets of states as its values, i.e., $f(x,a) = \{x_1, \ldots, x_k\}$, rather than just single values; here $x_1, \ldots, x_k \in X$ and $a \in A$. In contrast, the output function h takes only single values, i.e., there can be only one output character

associated with each state. The class of *deterministic finite state machines* is a subclass of non-deterministic finite state machines.

The operation of the machine Σ can be described as follows. The machine Σ starts from the initial state x_0 and accepts input sequences of the form $u := u_0 u_1 \ldots$, where $u_0, u_1, \ldots \in A$. In response to this set of applied input characters, it generates a sequence of states $x_0, x_1, x_2, \ldots \in X$ and a sequence of output values $y_0, y_1, y_2, \ldots \in Y$, according to the recursion in Equation 2-1.

Finally, the machine Σ is an *input/state machine* when $Y = X$ and the output is equal to the state at each step, that is,

$$y_k = x_k, k = 0, 1, 2, \ldots \tag{2-2}$$

An input/state machine Σ can be represented by the quadruple (A,X,x_0,f), since the output is the state.

Using the notion of the transition function f, we can define the partial function f^*: $X \times A^+ \rightarrow X$ by setting $f^*(x,u) := f(\ldots f(f(f(x,u_0),u_1),u_2)\ldots,u_k)$ for some state $x \in X$ and an input string $u = u_0u_1u_2\ldots u_k \in A^+$. Thus, f^* provides the value of the last state in the path generated by the input string u. For simplicity of notation, we will use f for f^*.

Some basic terminology important to the study of asynchronous machines is now described. A valid pair $(x,u) \in X \times A$ of the machine Σ is called a *stable combination* if $f(x,u) = x$, i.e., if the state x is a fixed point of the function f. An asynchronous machine will linger at a stable combination until an input change occurs. A pair (x,v) that is not a stable combination is called a *transient combination*. A *potentially stable state* x is one for which there is a stable combination. Of course, states that are not potentially stable serve only as transition states; the machine cannot stop or linger in

them, and they are not usable in applications. Due to this, it is common practice to ignore states that are not potentially stable; will adhere to this practice and not include such state in our state set.

Consider the sequential machine Σ of Equation 2-2. Let x be the state of the machine and u the input value. When a state-input pair (x,u) is not a stable combination, the machine Σ will go through a chain of input/state transitions, starting from (x,u), which may or may not terminate. If the machine reaches a stable combination (x',u) with the same input value, then the chain of transitions terminates. It must be noted that the input value must remain constant throughout the entire transition; and the transition will end if and only if such a stable combination as (x',u) exists. In such a case, x' is called the *next stable state* of x with input value u. When the next stable state is not uniquely determined, that is, the transition at some point has more than one value, we have a *critical race*. In case there is no next stable state for x with input value u, we obtain an *infinite cycle*. Critical races and infinite cycles are the types of hazards that can occur in an asynchronous machine.

2.2 Modes of Operation

When the value of an input variable is changed while the asynchronous machine undergoes a chain of transitions, the response of the machine may become unpredictable, since the state of the machine at which the input change occurs in unpredictable. To avoid this potential uncertainty, asynchronous machines are usually operated in *fundamental mode*, where only one variable of the machine is allowed to change each time. In particular, in fundamental mode operation, a change in the input variable is allowed only while the machine is in a stable combination.

One of the main topics of our present discussion relates to the control of an asynchronous machine in an infinite cycle. When a machine is in an infinite cycle, fundamental mode operation becomes impossible, since the machine will not reach a stable combination without a change of the input variable. Nevertheless, we will show later that, when taking the machine out of its infinite cycle, it is not necessary to forego fundamental mode operation for more than one step. When an asynchronous machine operates in fundamental mode in all but a finite number of steps, we say that it operates in *semi-fundamental mode*. The conditions and procedures for operation in semi-fundamental mode is described in Chapter 3.

Consider a defect-free asynchronous machine $\Sigma = (A,X,x_0,Y,f,h)$ operating in fundamental mode. Let the machine start at the state x_0, and be driven by the input string $w = v_0v_1\ldots v_{m-1}$. The machine now has transitions through the states $x_0,\ldots, x_m$, where $x_{i+1} = f(x_i,v_i)$; i = 0,1,2,…,m–1, where the last pair (x_m,v_{m-1}) is a stable combination. Fundamental mode operation implies that $v_i = v_{i+1}$ whenever (x_i,v_i) is not a stable pair, i = 0,…,m–1.

The *path* is defined as the set of pairs

$$P(\Sigma,x_0,w) = \{(x_0,v_0), (x_1,v_1),\ldots,(x_{m-1},v_{m-1}), (x_m,v_{m-1})\}. \qquad (2\text{-}3)$$

When a cycle occurs, there is no final stable pair, and the path becomes infinite

$$P(\Sigma,x_0,w) = \{\ldots,(x_i,v_j), (x_{i+1},v_j),\ldots,(x_{i+\ell},v_j), (x_i,v_j),\ldots\}.$$

As mentioned earlier, the step counter k advances one step during each state transition or input change. Of course, for fundamental mode operation, the input character remains constant during all transition chains.

Consider the case where the input value of the machine is kept constant at the character v, while the machine goes through a string of state transitions. At each state transition, the step counter of the system advances by one. This results in the input value being represented as a repetitive string vvv... v, where each repetition corresponds to an advance of the step counter. It is convenient to represent all such repetitions of the input character by a single character, so that, for example, a string of the form $w = v_0v_0v_1v_1v_1v_2v_2$ is represented by $w = v_0v_1v_2$. Here, it is understood that each input value is repeated as many times as necessary. In the case of an infinite cycle, the input value is repeated indefinitely.

Next, let Σ be an asynchronous machine without hazards. Then, the notion of the next stable state leads to the notion of the stable transition function, which plays an important role in our discussion. As the machine Σ has no hazards, every valid pair (x,u) of Σ has a next stable state x'. Define a partial function $s : X \times A \to X$ by setting $s(x,u) := x'$ for every valid pair (x,u), where x' is the next stable state of x with the input character u. The function s is called the *stable transition function* of the machine Σ. Note that the stable transition function describes the operation of the machine in the fundamental mode, i.e., under the requirement that the input value be changed only after the machine reaches a stable combination. This is a standard restriction in the operation of asynchronous machines, and it guarantees that the machine produces a deterministic response.

The stable transition function can be used to describe the operation of an asynchronous machine Σ in the following way. We know that the machine always starts from a stable combination (x_0,u_{-1}), at which the machine has been lingering for

some time. Suppose now that an input string $u := u_0u_1\ldots u_k \in A^+$ is applied to the machine. Then, the machine moves through the a list of states $x_1, \ldots, x_{k+1}$ of next stable states, where $x_{i+1} = s(x_i,u_i)$, which implies that (x_{i+1},u_i) is a stable combination for $i = 0, 1, \ldots, k$. The input list u is *permissible* if (x_i,u_i) is a valid combination for $i = 0, 1, \ldots, k$. Since the machine operates in the fundamental mode, the input character changes from u_i to u_{i+1} only after the stable combination (x_{i+1},u_i) has been reached. This guarantees that the state of the machine is well determined at the time when the input value changes.

In general, when a machine does not have infinite cycles, the intermediate transitory states are ignored, as they occur very speedily and are not noticeable by a user. Thus, the operation of an asynchronous machine without infinite cycles, as observed by a user, is best described by the stable transition function s. However, when the machine has infinite cycles, it is not possible to ignore the transitory states involved in the cycle, as the machine lingers among these transitions indefinitely. In Chapter 3, we will introduce a generalized notion of the stable transition function which accommodates infinite cycles by considering them as another form of persistent states.

Using the stable transition function s, we define the asynchronous sequential machine $\Sigma_{|s} := (A,X,Y,x_0,s,h)$, which is called the *stable state machine* induced by Σ. For an input/state machine, the stable state machine is given by the quadruple (A,X,x_0,s).

To deal with input strings, we can define a partial function $s^* : X\times A^+ \to X$ by setting $s^*(x,u) = s(\ldots s(s(s(x,u_0),u_1),u_2)\ldots,u_k)$, for some state $x \in X$ and a permissible input string $u = u_0u_1\ldots u_k \in A^+$. The partial function s^* provides the last state in the

path generated by the input string u. As before, we shall abuse the notation somewhat by using s for s^*.

In the case where we do not have a Moore machine, we can define the partial function $h^* : X \times A^+ \to Y$ in a similar fashion by setting $h^*(x,u) := h(\ldots s(s(s(x,u_0),u_1),u_2)\ldots,u_k)$ for a state $x \in X$ and a permissible input string $u = u_0u_1\ldots u_k \in A^+$, as was done for the stable transition function s defined earlier. The partial function h^* gives the last output value generated by the input list u.

It is necessary in some cases to compare different states of a sequential machine. Two states x and x' of a machine Σ are *distinguishable* if the following condition is valid: there is at least one finite input string which yields different output sequences when applied to Σ starting from the initial states x or x'. Two states x and x' of the machine Σ are *equivalent* when, for every possible input sequence, the same output sequence is produced regardless of whether the initial state is x or x'. Qualitatively, two states are equivalent if we cannot distinguish them by the input/output behavior of the machine. If two states are equivalent, then so are all corresponding states included in any paths started from x and x'. If a machine Σ contains two equivalent distinct states, then one of the states is redundant, and it can be omitted from the mathematical model of the machine. A machine Σ is *minimal* or *reduced* if it has no distinct states that are equivalent to each other.

Consider two machines $\Sigma = (A,Y,X,f,h)$ and $\Sigma' = (A',Y',X',f',h')$. Let x be a state of Σ and let x' be a state of Σ'. The states x and x' are *equivalent* if both have the same permissible input sequences, and if, for every permissible input sequence, the output sequence generated by Σ from the initial state x is the same as the output

sequence generated by Σ' from the initial state x'. The two machines Σ and Σ' are *equivalent* whenever there exists an equivalent state in Σ for every state in Σ', and similarly, an equivalent state in Σ for every state in Σ'. A number of texts dwell on the various aspects of the theory of sequential machines and automata, including McCluskey (1965); Arbib (1969); Kalman, Falb and Arbib (1969); Kohavi (1970); Eilenberg (1974); and Evans (1988).

CHAPTER 3
INFINITE CYCLES

3.1 Introduction

An infinite cycle is caused when there is no next stable state for a valid pair (x,u) of a finite state asynchronous machine $\Sigma = (A,X,x_0,Y,f,h)$. In such case, the machine will keep moving indefinitely from one transient combination to another. Only a finite number of states can be involved in these transitions, since Σ is a finite state machine. An infinite cycle can then be characterized by listing these states and the corresponding input which causes the infinite cycle. Specifically, let the state set of Σ be $X = \{x^1, x^2, ..., x^n\}$. Consider a infinite cycle ρ of Σ that involves p states, say the states $x^{k_1}, x^{k_2}, ..., x^{k_\ell} \in X$, and let $a \in A$ be the input character causing the infinite cycle. The infinite cycle then functions according to the recursion

$$x^{k_{j+1}} = f(x^{k_j}, a),\ j = 1, ..., \ell-1, \text{ and}$$
$$x^{k_1} = f(x^{k_\ell}, a). \qquad (3\text{-}1)$$

As a shorthand notation, we will use $\rho = \{a;x^{k_1}, x^{k_2}, ..., x^{k_\ell}\}$, where a is the input character of the infinite cycle, and $x^{k_1}, x^{k_2}, ..., x^{k_\ell}$ are its states. A state-input pair (x^{k_j},a), for some $j = 1, \ldots, \ell$ is then said to be a *pair of the cycle* ρ. The *length* ℓ of the infinite cycle ρ is defined as the number of distinct states it contains. It follows that a cycle of length ℓ has ℓ distinct input-state pairs. A brief examination of Equation 3-1 leads to the following conclusion: Let (x,a) be any pair of the infinite cycle ρ. Then, x'

is a state of ρ if and only if there is an integer j such that $x' = f^j(x,a)$. Then, the recursion can be followed for the entire length of the cycle. It is convenient to denote by $\{f^j(x,a)\}_{j=0}^{\ell}$ the set of all distinct states included in the set $\{x, f(x,a), f^2(x,a), ...\}$. This leads us to the following lemma.

Lemma 3-1 Let (x,a) be any pair of the infinite cycle ρ. Then, the state set of ρ is $\{f^j(x,a)\}_{j=0}^{\ell}$. ♦

Lemma 3-1 gives a way of finding the states of a cycle. A machine can have several infinite cycles, and we demonstrate below an algorithm that finds all infinite cycles of a given machine. It is relevant to see that an infinite cycle of length 1 is nothing but a stable combination. To distinguish between infinite cycles and stable combinations, the length of an infinite cycle must at least be 2. According to the next statement, infinite cycles associated with the same input character must have disjoint state sets.

Lemma 3-2 A valid input/state pair (x,a) of an asynchronous machine Σ can be a member of at most one infinite cycle.

Proof 3-2. Let ρ_1 and ρ_2 be two infinite cycles of the machine Σ associated with the same input character a. If ρ_1 and ρ_2 contain the same pair (x,a), then, using Lemma 3-1, we conclude that ρ_1 and ρ_2 have the same state set. In other words, $\rho_1 = \rho_2$, and our proof concludes. ♦

The above Lemma 3-1 gives us an insight into the structure of the cycle. Since no state can be a member of more than one cycle, we can say that a pair of a cycle is unique to that cycle. We can now find a bound on the maximal number of infinite cycles a

machine can have. Indeed, consider the set of all infinite cycles involving the input character a. In view of Lemma 3-2, all such infinite cycles have disjoint state sets. But then, the fact that a infinite cycle must contain at least 2 states, implies that there cannot be more than n/2 such infinite cycles, where n is the number of states of the machine. Finally, recalling that the machine has m input characters, we obtain the following. (Denote by $[a]^-$ the largest integer not exceeding a.)

Proposition 3-3 Let Σ be a sequential machine with n states and an alphabet A consisting of m characters. Then, Σ cannot have more than $m[n/2]^-$ infinite cycles. ♦

For the sake of brevity, "infinite cycles" will be referred to as just "cycles" in the rest of the document.

3.2 Detection of Cycles

In this section, a procedure to detect the cycles associated with a machine will be outlined. Consider an asynchronous machine $\Sigma = (A,X,x_0,Y,f,h)$ with n states and an alphabet A of m input characters. Next all the cycles of the machine Σ will be determined. To this end, an $n \times n$ matrix M(f) is recursively constructed as follows. The (i,j) entry of M(f) is the set of all characters $u \in A$ for which $x_i = f(x_j,u)$. If there is no input character $u \in A$ for which $x_i = f(x_j,u)$, then the (i,j) entry of M is denoted Γ, where Γ is a character not included in the alphabet set A. Thus, the matrix becomes

$$M_{i,j}(f) := \begin{cases} u \in A : x_i = f(x_j,u), \\ \Gamma \text{ otherwise,} \end{cases} \qquad (3\text{-}2)$$

i, j = 1, ..., n. Multiple entries in a cell are separated using a comma. We refer to M(f) as the *one-step transition matrix* of the machine Σ.

We now define two matrix operations for one-step transition matrices. First, the *sum* of two $n \times n$ one-step transition matrices A and B is defined element wise by

$$(A \cup B)_{ij} := A_{ij} \cup B_{ij},\ i, j = 1, ..., n. \tag{3-3}$$

The sum operation is reminiscent of the numerical addition of matrices. The following operations on members of the set $A \cup \Gamma$ is now defined.

$$\begin{aligned}
&\Gamma\Gamma := \Gamma \\
&u_1\Gamma := \Gamma \\
&\Gamma u_1 := \Gamma \\
&u_1u_1 := u_1 \\
&u_1u_2 := \Gamma \\
&\Gamma \cup \Gamma := \Gamma \\
&u_1 \cup \Gamma := u_1 \\
&u_1 \cup u_1 := \Gamma \\
&u_1 \cup u_2 := \Gamma
\end{aligned} \tag{3-4}$$

Also, for any two sets, the operation of *multiplication* is defined by

$$\{a_1, a_2, ..., a_q\}.\{b_1, b_2, ..., b_r\} = \{a_ib_j\}_{i=1, ...,q,\ j=1, ...,r}$$

This is similar to the classical definition of the scalar product of two vectors. Using these operations of multiplication, the definition of *matrix combination* is defined as follows.

$$(AB)_{ij} := \cup_{k=1, ..., n} A_{ik}B_{kj} \text{ for all } i, j = 1, ..., n. \tag{3-5}$$

Consider two functions $g, h : X \times A \to X$. The *composition* of g and h is defined by $gh(x,u) := g(h(x,u),u)$. The composition of function is closely related to the combination of the corresponding one-step transition matrices, as indicated by the following. The next statement shows that the multiplication of two one-step transition

matrices on two recursion functions is the transition matrix on the composition of the two functions.

Lemma 3-4 Let $g, h : X \times A \rightarrow X$ be two recursion functions with one step transition matrices $M(g)$ and $M(h)$, respectively. Then, $M(g)M(h) = M(gh)$.

Proof 3-4. Using Definition 3-4, we have

$$M(g)M(h) = \bigcup_{k=1..n} M_{i,k}(g)M_{k,j}(h). \qquad (3\text{-}6)$$

Now, each entry of $M(g)M(h)$ is either a character of A or Γ; the definition of matrix composition implies that the following two statements are equivalent:

(i) The (i,j) element of $M(g)M(h)$ is u.

(ii) There is a $k \in \{1, ..., n\}$ for which $M_{i,k}(g) = u$ and $M_{k,j}(h) = u$.

When (ii) holds, we have $x_i = g(x_k,u)$ and $x_k = h(x_j,u)$, so that $x_i = g(h(x_j,u),u) = gh(x_j,u)$. Then, the (i,j) entry of $M(gh)$ is u. Conversely, if the (i,j) entry of $M(gh)$ is u, then $x_i = gh(x_j,u)$, so that $x_i = g(h(x_j,u),u)$. Now, the value $h(x_j,u)$ is, by definition, an element of X, say the element x_k, $k \in \{1, ..., n\}$. Then, $x_k = h(x_j,u)$ and $x_i = g(x_k,u)$, which implies that (ii) is valid for this k. Thus, we have shown that (ii) is equivalent to

(iii) $M_{i,j}(gh) = u$.

Using the fact that (ii) is equivalent to (i), the Lemma follows. ♦

Thus, the product of two transition matrices with two different transition functions is the transition matrix of the composition of the two functions in the order of multiplication. Consider now the a machine Σ with recursion function f. Let $M(f)$ be

its one-step transition matrix. Then, by Lemma 3-4, $M(f)M(f) = M(f^2)$, $M(f^2)M(f) = M(f^3)$, and so on. This leads to the following corollary.

Corollary 3-5 Let $\Sigma = (A,X,x_0,Y,f,h)$ be an asynchronous machine. Then, for every integer $p > 0$, the one-step transition matrix satisfies $M^p(f) = M(f^p)$. ♦

Consider a machine $\Sigma = (A,X,x_0,Y,f,h)$. As mentioned earlier, a stable combination (x,u) of Σ can be considered as a cycle of length 1. This observation helps simplify the following statement, which forms an important tool in our process of finding all cycles of the machine Σ.

Lemma 3-6 Let Σ be an asynchronous machine with the recursion function f and the state set $X = \{x^1, ..., x^n\}$. Let $x^i \in X$ be a state and let u be an input character of Σ. Then, the following three statements are equivalent.

(i) There is an integer $\tau \geq 1$ such that $x^i = f^\tau(x^i,u)$

(ii) u appears on the (i,i) diagonal entry of the matrix $M^\tau(f)$.

(iii) (x^i,u) is a pair of a cycle whose length λ is an integer divisor of τ.

Proof 3-6. First, we show that (i) implies (ii). Assume that (i) is valid, so that $x^i = f^\tau(x^i,u)$ for some integer $\tau \geq 1$. Then, by Equation 3-2, the character u appears on the diagonal entry (i,i) of the matrix $M(f^\tau)$. By Corollary 3-5, $M(f^\tau) = M^\tau(f)$; this implies that u appears on the diagonal entry (i,i) of the transition matrix $M^\tau(f)$.

Next, we show that (ii) implies (iii). Let $\lambda \geq 1$ be the smallest integer satisfying $x^i = f^\lambda(x^i,u)$. Then, $\lambda \leq \tau$. Now, if $\lambda = 1$, then (x^i,u) is a stable combination of Σ, and hence forms a cycle of length 1. Otherwise, (x^i,u) is a pair of the cycle $\{x^i, f(x^i,u), ..., f^{\lambda-1}(x^i,u); u\}$ of length λ.

Now, using the integer division algorithm, we can write $\tau = q\lambda + \rho$, where q and ρ are integers and $0 \le \rho < \lambda$. If $\rho > 0$, then, using the equality $x^i = f^{\lambda}(x^i,u)$, we can write $x^i = f^{\tau}(x^i,u) = f^{\rho}(f^{q\lambda}(x^i,u),u) = f^{\rho}(x^i,u)$, so that $x^i = f^{\rho}(x^i,u)$. The latter contradicts the fact that λ was the smallest integer satisfying $x^i = f^{\lambda}(x^i,u)$, since $0 < \rho < \lambda$. Thus, we must have $\rho = 0$, and λ must be an integer divisor of τ.

Finally, we show that (iii) implies (i). Assume that (iii) is valid, i.e., that (x^i,u) is a pair of a cycle whose length λ is a divisor of τ. Then, there is an integer q such that $\tau = q\lambda$. But then, $f^{\tau}(x^i,u) = f^{q\lambda}(x^i,u) = x^i$, and (i) is valid. This concludes the proof. ♦

Thus, by taking the powers of the one-step transition matrix M(f) and observing the diagonal matrix at each step, the states of the machine involved in cycles can be isolated. The next lemma indicates a method of identifying the elements of each cycle.

Lemma 3-7 Let Σ be an asynchronous machine with state set $X = \{x^1, ..., x^n\}$ and recursion function f. Assume that Σ has a cycle of length $\lambda > 1$ that includes the pair (x^{j_0},u). Then, the other states $x^{j_1}, ..., x^{j_{\lambda-1}}$ of the cycle can be found from the matrix M(f) in the following way: for every integer $i = 0, ..., \lambda-2$, the index j_{i+1} is given by the position of the entry u in the column j_i of the matrix M(f).

Proof 3-7. Consider a cycle of length $\lambda > 1$. Let $\{x^{j_0}, x^{j_1}, ..., x^{j_{\lambda-1}}\}$ be the states involved in the cycle, and let u be the input character. We know $f(x^{j_0},u) = x^{j_1}$. This implies that the input character u appears exactly in the position (j_0, j_1) of the matrix M(f). Thus, the value of j_1 is given by the position of the character u in the row j_0. Continuing recursively, assume that, for some $0 \le k < \lambda$, the indices $j_0, j_1, ..., j_k$ of the states $x^{j_0}, x^{j_1}, ..., x^{j_k}$ of the cycle have been found. Then, since $x^{j_{k+1}} = f(x^{j_k},u)$, the

character u will appear exactly in position j_{k+1} of the row j_k of the matrix $M(f)$. This concludes our proof. ♦

Thus, the knowledge of the states involved in cycles of a particular length can be used to find the cycles specifically by applying Lemma 3-7. An algorithm that finds all the cycles of an asynchronous machine Σ whose recursion function is f will now be outlined. Let $\#A$ denote the number of elements in the set A.

Algorithm 3-8 Let $\Sigma = (A,X,x_0,Y,f,h)$ be an asynchronous machine, and let $M(f)$ be its transition matrix.

Step 1: All entries on the main diagonal of $M(f)$ represent stable combinations (i.e., cycles of length 1). For each input character $u \in A$, let $\Delta_1(u)$ be the set of all states for which u is included in a diagonal entry of the matrix $M(f)$; set $\vartheta_1(u) := \Delta_1(u)$.

Step 2: For $i \geq 2$, let $\Delta_i(u)$ be the set of all states for which u is included in a diagonal entry of the matrix $M^i(f)$. Define the difference set

$$\vartheta_i(u) := \Delta_i(u) \setminus \bigcup_{1 \leq j \leq i-1} \Delta_j(u). \tag{3-7}$$

Stop the algorithm for the character u when $i+1 > n - \Sigma_{j=1}^{i} \#\vartheta_j(u)$. ♦

The significance of Algorithm 3-8 is pointed out by the following statement.

Proposition 3-9 Let $i \geq 1$ be an integer. Then, in the notation of Algorithm 3-8, the following are true.

(i) The set $\vartheta_i(u)$ consists of all states of the machine Σ that are members of cycles of length i with the input character u.

(ii) The machine Σ has exactly $\#\vartheta_i(u)/i$ cycles of length i.

Suppose there are 2λ states in $\vartheta_\lambda(u)$. This means there are 2 cycles of length λ in the machine. Pick one state from $\vartheta_\lambda(u)$; use Lemma 3-7 to find the other $\lambda-1$ members of that cycle. Now, pick another state from the λ remaining states in $\vartheta_\lambda(u)$, and using Lemma 3-7, the second cycle can be determined. This procedure is extended to any number of cycles of a certain length.

Proof 3-9. Let $X = \{x^1, ..., x^n\}$ be the state set of Σ, and let $u \in A$ be a character of the input set. By Lemma 3-6, the following two statements are equivalent:

(a) The matrix $M^i(f)$ has an entry including u on its main diagonal, in a column that corresponds to the state x^r of Σ.

(b) The state x^r is a member of a cycle whose length is i or an integer divisor of i.

The case $i = 1$ is simple, since the elements on the main diagonal of $M(f)$ represent the stable combinations of the system, and whence the cycles of length 1. For $i > 1$, assume that the algorithm has not stopped before i, i.e., that $i \leq n - \Sigma_{j=1}^{i} \#\vartheta_j(u)$. Then, a slight reflection shows that the construction Equation 3-7 of $\vartheta_i(u)$ removes all states that are included in cycles of length less than i. Combining this with the equivalence of (a) and (b), it follows that $x^r \in \vartheta_i(u)$ if and only if x^r is a member of a cycle of length i.

Finally, consider the case where $i+1 > n - \Sigma_{j=1}^{i} \#\vartheta_j(u)$, and $\upsilon(u)$ be the number of states of Σ that are members of cycles of length $\lambda \leq i$ with the character u. From the definition of $\vartheta_j(u)$, it follows that $\upsilon(u) = \Sigma_{j=1}^{i} \#\vartheta_j(u)$. Thus, when $i + 1 > n - \upsilon(u)$, there are not enough states left to form a cycle of length $i + 1$ with the character u; whence

the machine Σ has no cycles of length greater than i with the character u. This completes the proof of part (i) of the Proposition.

Regarding part (ii) of the Proposition, note that the set $\vartheta_i(u)$ consists of all states that form cycles of length i with the input character u. By Lemma 3-2, no two cycles can have common states. Consequently, the number of elements of $\vartheta_i(u)$ is a multiple of i, and the number of cycles in $\vartheta_i(u)$ is given by $\#\vartheta_i(u)/i$. This concludes our proof.

♦

3.3 Stable State Representations for Machines with Cycles

In this section, a form of representation of only the next stable states and the extension of the definitions to the case of a machine with cycles will be explained. For an asynchronous machine $\Sigma = (A,X,Y,x_0,f,h)$ without cycles, a valid pair (x,u) always has a next stable state x'. Consequently, for such a machine, one can define a partial function $s : X\times A \rightarrow X$ by setting $s(x,u) = x'$ for every valid pair (x,u). The function s is then called the *stable recursion function* of the machine Σ. When s is used as a recursion function, it induces the *stable-state machine* $\Sigma_{|s} = (A,X,Y,x_0,s,h)$. The stable state machine describes only persistent states of the machine, and ignores unstable transitions. This function describes the behavior of Σ as experienced by a user, i.e., the transition of the machine to its next stable state is described. A reformulation of the finite state machine with cycles will now be explained, so that an equivalent stable state machine can be described from this reformulated machine.

As mentioned earlier in Chapter 2, asynchronous machines are normally operated in fundamental mode. In specific terms, fundamental mode operation is as follows. Let Σ be an asynchronous machine without cycles, resting in a stable combination (x,u_0).

Consider a string of input values $u = u_1 \ldots u_k \in A^+$ applied to the machine. When the input of Σ switches to u_1, the machine may engage in a string of transitions culminating in the next stable state $s(x,u_1)$. In fundamental mode operation, the input character u_1 is kept fixed until the machine has reached its next stable state, to guaranty deterministic behavior. Then, once the machine reaches the stable combination $(s(x,u_1),u_1)$, the input character is switched to u_2, and the machine eventually settles on its next stable state $s(s(x,u_1),u_2)$. This process continues until we reach the final input value u_k. The last stable state reached in this process is given by $x'' := s(\ldots(s(s(x,u_0),u_1),u_2)\ldots,u_k)$. To simplify our notation, we shall write in brief $x'' := s(x,u)$.

As mentioned earlier, the stable state machine describes persistent states of an asynchronous machine. For machines with cycles, a cycle describes a persistent state of the machine (i.e., the "state" of being in a cycle). Hence, when cycles are present, they must be represented in the stable state machine. Our next objective is to generalize the definition of the stable state machine, so it gives due representation to cycles, whenever they are present.

Consider an asynchronous machine $\Sigma = (A,Y,X,x_0,f,h)$ having $t > 0$ cycles ρ_1, ..., ρ_t. To define a stable state machine $\Sigma_{|s}$ that represents the persistent behavior of Σ, we first augment the state set X of Σ by t new elements x^{n+1}, ..., x^{n+t} to obtain the augmented state set

$$X^\rho := X \cup \{x^{n+1}, \ldots, x^{n+t}\}. \tag{3-8}$$

Each one of the new states, called the *cycle states,* corresponds to one of the cycles of Σ, i.e., x^{n+i} corresponds to the cycle ρ_i, $i = 1, \ldots, t$.

Define the function $s' : X \times A \to X^{\rho}$ by setting

$$s^1(x,u) = \begin{cases} s(x,u) \text{ if } x \in X \text{ and } (x,u) \text{ is a stable combination,} \\ x^{n+1} \text{ if } (x,u) \in \rho_i. \end{cases} \tag{3-9}$$

For a cycle $\rho = \{x^{k_1}, x^{k_2}, ..., x^{k_p};a\}$, $k_i \in [1,\ldots,n]$, denote

$$s^1[\rho,u] := \{s^1(x^{k_1},u), s^1(x^{k_2},u), ..., s^1(x^{k_p},u)\}. \tag{3-10}$$

Now, the function $s^2 : X^{\rho} \times A \to X^{\rho}$ is defined by setting

$$s^2(x,u) := \begin{cases} s^1(x,u) \text{ if } x \in X, \\ s^1[\rho_i,u] \text{ if } x = x^{n+i}. \end{cases} \tag{3-11}$$

The *generalized stable state machine* $\Sigma_{|s}$ of Σ is then the input/state machine defined by quintuple (A,X^{ρ},x_0,s^2). For the sake of notational convenience, we will omit the primes from s^2 in the future, and write $\Sigma_{|s} = (A,X^{\rho},x_0,s)$. The function s is called the *generalized stable recursion function* of Σ.

A cycle ρ associated with a cycle state x is said to be *stoppable* if there exists an input character $u \in A$ such that at least one of the outcomes of $s(x,u) \in X$. In the present work, we assume that all the cycles are stoppable. It is necessary to make this assumption; if not, there will be no controller that can stop the cycle.

Note that when there are no cycles, the generalized stable recursion function of Σ is identical to the stable recursion function of Σ. Let $\Sigma_{c|s}$ be the generalized stable state machine induced by the closed loop system Σ_c shown in Figure 1-1. The present work concentrates on finding the solution to the next statement; consequently, this statement is the main topic of our discussion.

The Model Matching Problem 3-10 Let Σ be a machine and let Σ' be a stable state machine with the same input and output alphabets as Σ. Find necessary and sufficient conditions for the existence of a controller C such that the generalized stable state machine $\Sigma_{c|s} = \Sigma'$. When C exists, provide a method for its design. ♦

The controller C of Problem 3-10 assigns to closed loop system the stable state behavior of the specified machine Σ'. Of particular interest to us here is the case where the machine Σ has infinite cycles. In such case, the controller C of 3-10 eliminates the effect of infinite cycles on the closed loop system. The model matching problem for asynchronous machines with critical races, but not infinite cycles, was discussed in Murphy 1996, Murphy, Geng and Hammer 2002, 2003, Geng 2003- In the following section, a solution for the model matching problem for the case where the controlled system Σ has infinite cycles is derived, and in Chapter 5 the design of an appropriate controller is demonstrated by means of an example.

3.4 Stable Reachability

The extension of the stable state machine to machines with cycles will now be described. This requires a reformulation of the machine. Consider an asynchronous machine Σ represented by the sextuple (A,Y,X,x_0,f,h), and assume that Σ has $t > 0$ cycles $\rho_1, ..., \rho_t$. Let $\Sigma_{|s} = (A,X^\rho,x_0,s)$, where $X^\rho = X \bigcup \{x^{n+1}, ..., x^{n+t}\}$, and x^{n+i} is the generalized state corresponding to the cycle ρ_i. We aim to define the notion of stable reachability for the machine Σ so as to characterize the "stable" states Σ can reach from a given initial condition, in response to various input strings. To this end, we need to define several notions. A *generalized stable combination* of the machine Σ is either a

stable combination of Σ or a pair of the form (x^{n+i},u), where u is the input character of the cycle ρ_i. A state of $\Sigma_{|s}$ is *potentially stable* if it is part of a generalized stable combination of Σ.

When applying a control input to a machine that is in an cycle, it is not possible to operate in the fundamental mode. The input character must be changed while the cycle is in progress in order to attempt to interrupt the cycle. In other words, the input character is changed while the machine is not in a stable combination. Of course, if this change in input leads the machine to a stable combination, then fundamental mode operation can be resumed from there on. We arrive at the following notion, which describes the mode of operation that is closest to fundamental mode operation.

Definition 3-11 A machine Σ is said to operate in *semi-fundamental mode* if the machine operates in fundamental mode when it is not in an cycle. ♦

Recall that the states of the generalize stable-state machine $\Sigma_{|s}$ are either potentially stable states of Σ or cycle state that represent cycles. Thus, semi-fundamental mode operation of Σ becomes fundamental mode operation of $\Sigma_{|s}$.

Definition 3-12 Let Σ be an asynchronous machine with generalized state set X^ρ and generalized stable recursion function s. A state $x' \in X^\rho$ is *stably reachable* from a state $x \in X^\rho$ if there is a input string $u = u_0u_1\ldots u_k$ of Σ for which $x' = s(x,u)$. ♦

The following assumption is essential in determining the controller operation. If an input character forms a valid combination with one outcome of a critical race, then it forms a valid combination with all other outcomes of the same race. This assumption can always be satisfied by extending the definition of the recursion function of the machine.

It is necessary to assume this statement because we need the controller to be operational will all outcomes of the critical race that follows the cycle.

3.5 Matrix of Stable Transitions and the Skeleton Matrix

Consider an asynchronous machine $\Sigma = (A,Y,X,x_0,f,h)$ with the state set $\{ x^1, ..., x^n\}$, and assume that Σ has $t > 0$ cycles $\rho_1, ..., \rho_t$. Let $\Sigma_{|s} = (A,X^\rho,x_0,s)$, where $X^\rho = X \bigcup \{x^{n+1}, ..., x^{n+t}\}$ and x^{n+i} is the generalized state corresponding to the cycle ρ_i. Let $s^x(x^j,x^i)$ be the set of all characters $u \in A$ such that $x^i \in s(x^j,u)$. The *matrix of one-step generalized stable transitions* gives all the possible stable one-step transitions of the machine $\Sigma_{|s}$; it is an $(n+t)\times(n+t)$ matrix $R(\Sigma_{|s})$, whose (i,j) entry is given by

$$R_{ij}(\Sigma_{|s}) = \begin{cases} s^x(x^j,x^i) \text{ if } s^x(x^j,x^i) \neq \varnothing, \\ N \text{ otherwise,} \end{cases} \tag{3-12}$$

$i, j = 1, ..., n+t$. Recall that, if x^j is a generalized state, then a critical race may occur when an input character u is applied at x^j. This race is represented in the matrix $R(\Sigma_{|s})$ by the appearance of the character u in more than one row of column j. Note that Equation 3-12 is similar to Equation 3-2, except for the fact that $R(\Sigma_{|s})$ has some augmented states instead of cycles. We turn now to a description of some operations on the matrix $R(\Sigma_{|s})$.

Let A^* be the set of all words over the alphabet A. The operation of *unison* $\bigcup\!\!\!\!\!\cup$ is defined over the set $A^* \bigcup N$ as follows. Let w_1, w_2 be either subsets of the set A^* or the character N; then, set

$$w_1 \veebar w_2 := \begin{cases} w_1 \cup w_2 \text{ if } w_1 \subset A^* \text{ and } w_2 \subset A^*, \\ w_1 \text{ if } w_1 \subset A^* \text{ and } w_2 = N, \\ w_2 \text{ if } w_1 = N \text{ and } w_2 \subset A^*, \\ N \text{ if } w_1 = w_2 = N. \end{cases} \tag{3-13}$$

Extending this definition to matrices, the *unison* $C := A \veebar B$ of two $n \times n$ matrices A and B is defined by $C_{ij} := A_{ij} \veebar B_{ij}$, $i, j = 1, ..., n$, i.e., an element-wise operation as defined in Equation 3-13-

As usual, the concatenation of two words $w_1, w_2 \in A^*$ is given by $w_1 w_2$, i.e., w_1 followed by w_2. For elements w_1, w_2 are elements of A^* or the character N, the *concatenation* is defined as follows.

$$\text{conc}(w_1,w_2) := \begin{cases} w_1 w_2 \text{ if } w_1, w_2 \in A^*, \\ N \text{ if } w_1 = N \text{ or } w_2 = N. \end{cases} \tag{3-14}$$

More generally, let $W = \{w_1, w_2, \ldots, w_q\}$ and $V = \{v_1, v_2, \ldots, v_r\}$ be two subsets, whose elements are either words of A^* or the character N. Define

$$\text{conc}(W,V) := \veebar_{\substack{i=1,\ldots,q \\ j=1,\ldots,r}} \text{conc}(w_i,v_j). \tag{3-15}$$

Note that the concatenation is either a subset of A^*, or it is the character N. The concatenation is non-commutative, and N takes the place of a "zero".

Let C and D to be two $n \times n$ matrices whose entries are subsets of A^* or the character N. Let C_{ij} and D_{ij} be the (i,j) entries of the corresponding matrices. Then, the *product* $Z := CD$ is an $n \times n$ matrix, whose (i,j) entry Z_{ij} is given by

$$Z_{ij} := \veebar_{k=1}^{n} \text{conc}(C_{ik},D_{kj}), \; i,j = 1, \ldots, n. \tag{3-16}$$

Using this notation, we can define powers of the one-step transition matrix recursively, by setting

$$R^{\mu}(\Sigma_{|s}) := R^{\mu-1}(\Sigma_{|s})R(\Sigma_{|s}),\ \mu = 2, 3, \ldots \tag{3-17}$$

The matrix $R^{\mu}(\Sigma_{|s})$ has the following physical significance. The (i,j) element of $R^{\mu}(\Sigma_{|s})$ is the set of all input strings that take x^j to x^i in exactly μ steps. Thus, $R^{\mu}(\Sigma_{|s})$ is called the *matrix of μ-step stable transitions*. This follows from Corollary 3-5, which states that the powers of the one-step transition matrix is equal to the transition matrix of the powers of the recursion function, i.e., the steps of the transition matrix.

Define the matrix

$$R^{(\mu)}(\Sigma_{|s}) := \bigcup_{k=1\ldots\mu} R^{k}(\Sigma_{|s}),\ v=2,3,...,(n+t-1). \tag{3-18}$$

By construction, the (i,j) entry of $R^{(\mu)}(\Sigma_{|s})$ consists of all strings that can take the machine $\Sigma_{|s}$ from the state x^j to the state x^i in μ or fewer steps. The following statement is important to our discussion; the proof is similar to that of Murphy, Geng and Hammer, 2003, Lemma 3-9.

Lemma 3-13 The following two statements are equivalent:

(i) There is an input string that can take the machine $\Sigma_{|s}$ from the state x^j to the state x^i.

(ii) The i,j entry $R_{ij}^{(n+t-1)}(\Sigma_{|s}) \neq N$. ♦

Now, $R^{(n+t-1)}(\Sigma_{|s})$ contains not only the stable transitions, but also the critical races that are cause when an input character is applied to a cycle state. To satisfy the model matching problem, we need a matrix for which the same input string does not lead to

multiple outcomes. This is achieved by modifying the above matrix. This modified matrix is at the center of our discussion.

Definition 3-14 Let Σ be an asynchronous machine. The *matrix of stable transitions* $T(\Sigma_{|s})$ is constructed by performing the following operation on each column of the matrix $R^{(n+t-1)}(\Sigma_{|s})$: remove all occurrences of strings that appear in more than one entry of the column; if empty entries result, replace them by the character N. ♦

To understand the significance of the matrix $T(\Sigma_{|s})$, it is instructive to compare it to the matrix $R^{(n+t-1)}(\Sigma_{|s})$. A string u in the (i,j) entry of the matrix $R^{(n+t-1)}(\Sigma_{|s})$ indicates, of course, that u takes $\Sigma_{|s}$ from x^j to x^i. However, if the string u appears in another entry in column j of $R^{(n+t-1)}(\Sigma_{|s})$, say in entry k, then the same string may also take $\Sigma_{|s}$ from x^j to x^k; this indicates a critical race. The construction of the matrix $T(\Sigma_{|s})$ removes then all transitions that involve critical races. As a result, an input string u that is included in the (i,j) entry of $T(\Sigma_{|s})$ takes the machine $\Sigma_{|s}$ from the state x^j to the state x^i, and to no other state.

Definition 3-15 Let Σ be an asynchronous machine with the generalized state set $\{x^1, \ldots x^{n+t}\}$. A generalized stable transition from a generalized state x^j to a generalized state x^i is a *uniform transition* when one of the following is true: (i) x^i can be reached from x^j without passing through a critical race, or (ii) there is a succession of critical races through which x^i can be reached from x^j for all possible race outcomes. ♦

Note that in case (ii) of Definition 3-15, the input string applied to the machine may depend on the outcomes of the critical races encountered by the machine along its way from the state x^j to the state x^i. Specifically, let x(k,p) be outcome k of race p along

the way from x^j to x^i. The input string applied to the machine will then be a function of the outcomes $u(x(k_{1,1},1), ..., x(k_{1,p_1},p_1))$. The set $\bigcup_{a=1,\ldots,r} u(x(k_{r,1},1), ..., x(k_{r,p_r},p_r))$ is *(i,j) transition set*, where r is the number of possible input strings taking into account all the critical races encountered in the path from x^j to x^i. This means that p_r races are encountered in the path followed by the rth string, and $x(k_{r,p_r},p_r)$ is the outcome k_{r,p_r} of the race p_r. Further, By construction, the elements of $T(\Sigma_{|s})$ will contain sets of strings that satisfy the following proposition.

Proposition 3-16 The (i,j) entry of $T(\Sigma_{|s})$ is different from N if and only if there is a uniform transition from x^j to x^i. Furthermore, if the (i,j) entry of $T(\Sigma_{|s})$ is not N, then it includes an (i,j) transition set. ♦

Thus, depending on the sequence of critical races that occur, the entries of $T(\Sigma_{|s})$ gives a set of strings that can be used for the control process. The skeleton matrix is now defined as follows matrix:

Definition 3-17 Let $T(\Sigma_{|s})$ be the matrix of stable transitions of the generalized stable state machine $\Sigma_{|s}$. The *skeleton matrix* $K(\Sigma)$ of Σ is a matrix of zeros and ones, whose (i,j) entry is given by

$$K_{ij}(\Sigma) = \begin{cases} 1 \text{ if } T_{ij}(\Sigma_{|s}) \neq N, \\ 0 \text{ otherwise} \end{cases} \qquad (3\text{-}19)$$

$i = 1,\ldots,n,\ j = 1,\ldots,n+t.$ ♦

The skeleton matrix $K(\Sigma)$ has a simple interpretation. In view of Lemma 3-13, the following is true

Proposition 3-18 Let $K(\Sigma)$ be the skeleton matrix of a given machine Σ. Then, the (i,j) entry of $K(\Sigma)$ is 1 if and only if there exists a string that takes the machine $\Sigma_{|s}$ from the generalized state x^j to the state x^i and to no other state. ♦

The skeleton matrix plays a critical role in our discussion, reminiscent of the role it played in Murphy, Geng and Hammer 2002, 2003-

3.6 Corrective Controllers

The basic considerations in the construction of a corrective controller and its limitations for asynchronous input/state machines will now be outlined. Let Σ be an asynchronous machine, being controlled in the configuration Figure 1-1 with the controller C. Recall that Σ_c denotes the closed loop machine of Figure 1-1; let $\Sigma_{c|s}$ be the stable state machine induced by Σ_c. Now, let $\Sigma' = (A,X,x_0,s')$ be a stable state input/state machine, with the same input set and the same state set as the machine Σ. Referring to the configuration Figure 1-1 and the Model Matching Problem 3-10, we are investigating the existence of a controller C for which $\Sigma_{c|s} = \Sigma'$.

Consider the input/state machine $\Sigma = (A,X,X,f,h)$ and the controller $C = (A\times X,A,\Xi,\phi,\eta)$. Let the generalized stable state machine associated with Σ be $\Sigma_{|s}$. As shown below, the controller C will operate in the semi-fundamental mode: when the controlled machine Σ moves toward a stable combination, the controller C will operate in fundamental mode - it will keep its output value constant until Σ has reached its next stable combination; however, if Σ moves into an infinite cycle, the controller has no choice but to change its output value while Σ is cycling - otherwise, Σ will remain in the cycle forever. In this case, the controller operates in the semi-fundamental mode.

Recalling that Σ has the state set X and that C has the state set Ξ, it follows that the composite machine Σ_c of Figure 1-1 has the state set $X\times\Xi$. We can then represent Σ_c by the quintuple $(A,X,X\times\Xi,f_c,h_c)$, where f_c is the recursion function of Σ_c and h_c is its output function. Being an input/state system, the machine Σ has the output function $h(x,u) = x$. As the output of Σ_c is the same as the output of Σ, it follows that $h_c(x,\xi,v) = x$. Formally, let $\pi_x : X\times\Xi\rightarrow X : \pi_x(x,\xi) \mapsto x$ be the standard projection. Then, we have $h_c = \pi_x$.

Now, let γ be the recursion function of $\Sigma_{c|s}$. We know that the output of $\Sigma_{c|s}$ is the state of Σ. So, if $\Sigma_{c|s} = \Sigma'$, then, for every valid pair (x,v) of Σ', there is a state $\xi \in \Xi$ for which (x,ξ,v) is a valid pair of $\Sigma_{c|s}$. This leads to the conclusion

$$h_c\gamma(x,\xi,v) = s'(x,v), \tag{3-20}$$

and, using the equality $h_c = \pi_x$, we obtain $\pi_x\gamma(x,\xi,v) = s'(x,v)$. Note that, if s' is not constant over one of the cycles, then the response of the closed loop machine will depend on the state of the cycle at which the input character is applied, creating a critical race. In other words, in order for the response of the closed loop system to be deterministic, the stable recursion function of the model must be constant over each cycle of the controlled machine Σ. To state this fact formally, consider a cycle $\rho = \{x^{k_1}, x^{k_2}, ..., x^{k_p};a\}$, $k_i \in [1,\ldots,n]$, of the machine Σ. Let $s'\{\rho\} := \{s'(x^{k_1},a), ..., s'(x^{k_p},a)\}$, i.e., the set of values of s' over the cycle. We say that s' is *constant over the cycle* ρ if all members of the set $s'\{\rho\}$ are identical.

Proposition 3-19 Let Σ and Σ' be two machines, where Σ' is a stable state machine. Assume that there is a controller C such that $\Sigma_{c|s} = \Sigma'$. If the closed loop machine $\Sigma_{c|s}$ has a race free response, then the recursion function s' of the model Σ' is constant over every cycle of Σ. ♦

We are now ready to state one of the main results of our discussion. The following theorem deals with the fundamental conditions for the existence of a model matching controller (compare to Murphy, Geng and Hammer 2002, 2003, Theorem 4.3). Before stating the theorem, we need some notation. For a cycle state $z \in X^\rho$, let $\rho(z) = \{a; x^{k1}, x^{k2}, ..., x^{kp}\}$, $k_i \in [1,\ldots,n]$, be the corresponding cycle, and let $\pi_x\rho(z) := \{x^{k1}, x^{k2}, ..., x^{kp}\}$ be the states of the cycle. Define

$$\varphi(z) = \begin{cases} z \text{ if } z \text{ is a regular state,} \\ \pi_x\rho(z) \text{ if } z \text{ is a cycle state.} \end{cases} \tag{3-21}$$

Note that $\varphi(z)$ is always a set of regular states, not cycle states.

Theorem 3-20 (*Existence of Controller*) Let $\Sigma_{|s} = (A,X^\rho,x_0,s)$ be the generalized stable state machine induced by an input/state machine $\Sigma = (A,X,x_0,f)$. Let $z^1, ..., z^k$ be generalized states of $\Sigma_{|s}$ and let $U_1, ..., U_k$ be sets of input characters such that $z^1\times U_1$, $z^2\times U_2, \ldots, z^k\times U_k$ are distinct sets of valid pairs of $\Sigma_{|s}$. For each $i = 1, ..., k$, let z'^i be a state of Σ that is stably reachable from the generalized state z^i. Then, there exists a controller C for which $\Sigma_{c|s}$ is equivalent to a stable state machine $\Sigma' = (A,X,x_0,s')$, whose recursion function s' satisfies

(i) $s'[\varphi(z^i),U_i] = z'^i$ for all $i = 1,\ldots,k$, and

(ii) $s'(x,t) = s(x,t)$ for all $(x,t) \in X\times A / \bigcup_{i=1,\ldots,k} z^i\times U_i$.

Moreover, the closed loop system Σ_c is well posed, and it operates in the semi-fundamental mode.

Proof 3-20. By Proposition 3-19, the recursion function of the model is constant over all cycles of the machine Σ. Thus, since z'^i is stably reachable from the generalized state z^i, there exists a string $w_i \in A^+$ such that $s(z^i,w_i) = z'^i$, where s is the stable recursion function of Σ. Let $m(i)$ be the length of w_i; write $w_i = v_i^0 v_i^1 \ldots v_i^{m(i)-1}$, where $v_i^0, v_i^1, \ldots, v_i^{m(i)-1}$ are individual characters of the input alphabet A and $i = 1, \ldots, k$. With this input string w_i, the recursion function s generates a string of generalized states, which we denote by

$$\begin{aligned} &x^{i,1} := s(z^i,v_i^0),\\ &x^{i,2} := s(x^{i,1},v_i^1), \ldots,\\ &x^{i,m(i)-1} := s(x^{i,m(i)-2},v_i^{m(i)-2}),\\ &z'^i := s(x^{i,m(i)-1},v_i^{m(i)-1}),\ i = 1, \ldots, k. \end{aligned} \tag{3-22}$$

Let $U(z^i) \subset A$ be the set of all input characters that form stable combinations with the generalized state z^i, $i = 1, \ldots, k$. Define the following sets:

$$\begin{aligned} &S := \bigcup_{i=1, \ldots, k} z^i \times U(z^i),\\ &V := \bigcup_i = 1, \ldots, k\ z^i \times U_i, \end{aligned} \tag{3-23}$$

where U_i are the input character sets given in the statement of the Theorem. Then, a controller $C = (A\times X, A, \Xi, \phi, \eta)$ that satisfies the requirements of Theorem 3-20 can be constructed as follows.

(i) The state set Ξ of C has $2 + \Sigma_{i=1}^{k} m(i)$ states given by

$$\Xi = \{\xi_0, \xi_1, \xi_1^1,\ldots, \xi_1^{m(1)}, \xi_2^1,\ldots, \xi_2^{m(2)},\ldots, \xi_k^1, \ldots, \xi_k^{m(k)}\}. \tag{3-24}$$

The significance of these states is explained below.

(ii) The initial state of the controller C is ξ_0. The controller moves to the state ξ_1 upon the detection of a stable combination with one of the generalized states $z^1,\ldots, z^k \in X^\rho$. Note that, for cycle states, this transition occurs during the cycle, and is therefore a semi-fundamental mode transition. For states that are not cycle states, this transition is in fundamental mode. To implement these transitions, the recursion function ϕ of C is defined by

$$\begin{aligned}&\phi(\xi_0,(z,t)) := \xi_0 \text{ for all } (z,t) \in X\times A \setminus S,\\ &\phi(\xi_0,(x,u)) := \xi_1 \text{ for all } (x,u) \in S.\end{aligned} \tag{3-25}$$

The state ξ_0 indicates that the machine Σ is not in a state in which its response must be changed, and hence, when in the state ξ_0, the controller does not interfere with the operation of Σ; it simply applies to Σ its own external input. To this end, the output function η of C at the state ξ_0 is defined by

$$\eta(\xi_0,(z,t)) := t \text{ for all } (z,t) \in X\times A. \tag{3-26}$$

The output function at the state ξ_1 is defined as follows. Choose a character $u_i \in U(z^i)$; set

$$\eta(\xi_1,(z^i,t)) := u_i \text{ for all } t \in A,\ i = 1, \ldots, k. \tag{3-27}$$

Then, the machine Σ continues to rest at the generalized state z^i.

(iii) Next, assume that Σ is in a stable combination with the generalized state z^i, $i = 1, \ldots, k$, and an input value $u \in U_i$ is applied. Then, the controller C will begin to

apply the string w_i to Σ, to take Σ to the state z'_i. To this end, the recursion function of C is defined as follows.

$$\phi(\xi_1,(z^i,u)) := \xi_i^1 \text{ for all } u \in U_i,\ i = 1, \ldots, k;$$
$$\phi(\xi_1,(z,t)) := \xi_0 \text{ for all pairs } (z,t) \in X\times A\backslash V. \tag{3-28}$$

Upon reaching the state ξ_i^1, the controller generates the first character of the input string w_i of Σ, to start taking Σ to the desired state z'^i. Accordingly, the controllers output function value at this point is

$$\eta(\xi_i^1,(z,t)) = v_i^0 \text{ for all } (z,t) \in X\times A,\ i = 1, \ldots, k. \tag{3-29}$$

Upon application of this input character, the machine Σ will move to its next generalized stable state $x^{i,1}$, where $(x^{i,1}, v_i^0)$ is the next generalized stable combination.

(iv) In order for the controller to continue to generate the string w_i as input for Σ, we define its recursion function as follows.

$$\phi(\xi_i^j,(x^{i,j},u)) := \xi_i^{j+1} \text{ for all } u \in U_i, \tag{3-30}$$

$j = 1,\ldots,m(i)-1$, $i = 1, \ldots, k$. Now, the controller feeds to Σ the next character in the string w_i, i.e., the character v_i^j. So the output function must satisfy

$$\eta(\xi_i^{j+1},(z,u)) := v_i^j \text{ for all } (z,u) \in X\times A, \tag{3-31}$$

$i = 1,\ldots,k$, $j = 1, 2, \ldots, m(i)-1$.

(v) At the step $j = m(i)-1$, the controller completes generating the string w_i. It assumes then a stable combination, and continues to feed the input character $v_i^{m(i)-1}$ until its external input character is changed. To implement these requirements, the transition and output functions of the controller are defined by

$$\phi(\xi_i^{m(i)},(z'^i,u)) := \xi_i^{m(i)} \text{ for all } u \in U_i$$
$$\phi(\xi_i^{m(i)},(z,t)) := \xi_0 \text{ for all } (z,t) \in X\times A\setminus z'^i\times U_i,$$
$$\eta(\xi_i^{m(i)},(z,u)) := v_i^{m(i)-1}. \tag{3-32}$$

$i = 1,\ldots,k$. Note that, since the state-input sets $\{z^i\times U_i\}_{i=1}^{k}$ are all disjoint, there is no ambiguity in the definition of the recursion function ϕ.

From the above construction, the stable recursion function γ of the closed loop system Σ_c can be deduced. On analysis, this recursion function satisfies, for all $i = 1,\ldots,k$:

$$\gamma(z^i,\xi_0,u) = (s(z^i,u),\xi_1) \text{ for all } u \in U(z^i),$$
$$\gamma(z,\xi_0,t) = (s(z,t),\xi_0) \text{ for all } (z,t) \in X\times A\setminus(S\cup V),$$
$$\gamma(z^i,\xi_1,u_i) = (z'^i,\xi_i^{m(i)}) \text{ for all } u_i \in U_i,$$
$$\gamma(z^i,\xi_1,u) = (z^i,\xi_1) \text{ for all } u \in U(z^i),$$
$$\gamma(z^i,\xi_1,t) = (z^i,\xi_0) \text{ for all } t \in A\setminus(U(z^i) \cup U_i),$$
$$\gamma(z'^i,\xi_i^{m(i)},u) = (z'^i,\xi_i^{m(i)}) \text{ for all } u \in U_i,$$
$$\gamma(z,\xi_i^{m(i)},t) = (s(z,t),\xi_0) \text{ for all } (z,t) \in X\times A\setminus z'^i\times U_i, \tag{3-33}$$

$i = 1,\ldots,k$. Thus, the equivalence of $\pi_x\gamma$ and s' as defined in the beginning of this section holds true. Also, the closed loop system operates in the semi-fundamental mode as the definitions of ϕ and η specify. We know Σ is an input-state system, hence it is strictly causal; consequently the closed loop system Σ_c is well posed (Hammer 1996). This concludes the proof. ♦

Chapter 4 deals with the specific solution to the model matching problem for this problem. The necessary and sufficient conditions for the existence of the controller, using the skeleton matrices are put forward, and if the controller exists, an algorithm to construct the states is also given.

CHAPTER 4
THE MODEL MATCHING PROBLEM

In this section we present a solution to the model matching problems for machines with infinite cycles. The solution to the model matching problem for deterministic systems and for machines with races was presented in Murphy, Geng and Hammer, (2002, 2003). Let Σ be an asynchronous machine with all cycles stoppable, and let Σ' be the stable-state model to be matched. The skeleton matrix $K^0(\Sigma')$ of the model Σ' is given as in Definition 3-17. For the solution to the model matching problem, a form of the skeleton matrix with columns augmented to include the solution to cycles is introduced; this is the *augmented skeleton matrix* $K(\Sigma')$.

The matrix $K(\Sigma')$ is an $n\times(n+t)$ matrix of zeros and ones. It is obtained by augmenting the matrix $K^0(\Sigma')$ with t columns corresponding to the cycle states, as follows. Let z be a cycle state, and let $\pi_x\rho(z)$ be the states of the cycle. Then,

$$K_{ij}(\Sigma') = \begin{cases} K^0_{ij}(\Sigma'),\ i,j = 1,...,n, \\ 1 \text{ if } K^0_{ij}(\Sigma') = 1 \text{ for all } x^k \in \pi_x\rho(x^j),\ i = 1,...,n,\ j = n+1,...,n+t, \\ 0 \text{ otherwise.} \end{cases} \quad (4\text{-}1)$$

In view of Proposition 3-19, the model Σ' can be matched only if its stable transition function is constant over the states of each cycle of Σ. Assume then that Σ' satisfies this requirement. In such case, it follows by Equation 4-1 that the entries of the column corresponding to x^k, $k = n+1,\ldots,n+t$ in $K(\Sigma')$ are all 1. The solution to the model matching problem for a machine with infinite cycles can now be stated as follows.

Theorem 4-1 Let Σ be an asynchronous machine all of whose cycles are stoppable, and let $\Sigma_{|s} = (A,X^{\rho},x^0,s)$ be the generalized stable state machine induced by Σ. Let $\Sigma' = (A,X,x^0,s')$ be a stable-state input/state machine with the same state set X and the same input set A as those of Σ. Let $K(\Sigma)$ be the skeleton matrix of Σ, and let $K(\Sigma')$ be the skeleton matrix of Σ'. Then, the two following statements are equivalent.

(i) There is a controller C for which the closed loop system $\Sigma_{c|s}$ is equivalent to Σ', where $\Sigma_{c|s}$ operates in the semi-fundamental mode and is well posed.

(ii) The skeleton matrices satisfy $K(\Sigma) \geq K(\Sigma')$.

Theorem 4-1 provides a simple necessary and sufficient condition for the existence of a solution to the model matching problem for systems with infinite cycles. The proof of the Theorem, which is provided later in this section, includes an algorithm for the construction of an appropriate controller C. When the model matching problem is solvable, the controller C operates by transforming into unstable combinations all generalized stable combinations of Σ that do not correspond to stable combinations of the model Σ'. In this way, the stable state machine $\Sigma_{c|s}$ induced by the closed loop system becomes stably equivalent to Σ'.

The skeleton matrix of the given machine $K(\Sigma)$ is never fully zeros, since each state is potentially stable. Thus, there always exists a stable state machine Σ'' with skeleton matrix equal to $K(\Sigma)$. When considered as a model, the machine Σ'' satisfies condition (ii) of Theorem 4-1. Consequently, there is a controller C for which $\Sigma_{c|s} = \Sigma''$, and this controller eliminates the effects of the infinite cycles of Σ. Thus, feedback

controller can turn every machine with stoppable cycles into a deterministic stable state machine.

We turn now to a preliminary technical discussion that will lead us to the proof of Theorem 4-1. Let s be the stable recursion function of the generalized stable state machine $\Sigma_{|s}$ of Σ, and let s' be the stable recursion function of the model Σ', with the co-domain of s' extended to X^{ρ}. Build the *discrepancy set* of Σ as

$$D(\Sigma,\Sigma') := \{(x,u) \in X\times A : (x,u) \text{ is a valid pair of } \Sigma' \text{ and } s(x,u) \neq s'(x,u)\}. \tag{4-2}$$

In intuitive terms, $D(\Sigma,\Sigma')$ is the set of all valid pairs of Σ' for which the next stable state of Σ' is different from the next generalized state of $\Sigma_{|s}$.

Now, since $K(\Sigma) \geq K(\Sigma')$, for each $(x,u) \in D(\Sigma,\Sigma')$, there exists an input string $w \in A^+$ for which $s(x,w) = s'(x,u)$; let

$$S(x,u) \subset A^+ \tag{4-3}$$

be the set of all input strings w satisfying $s(x,w) = s'(x,u)$. Next, consider a pair $(x,u) \in D(\Sigma,\Sigma')$ and an input string $w \in S(x,u)$. Let $P(\Sigma,x,w)$ be the corresponding path of Σ, and let

$$\rho(\Sigma) := \{(z,r) : (z,r) \in X\times A, (z,r) \in \rho_i, i = 1,\ldots,t\} \tag{4-4}$$

be the set of all state-input pairs involved in cycles. Define the set

$$D_N(\Sigma,\Sigma') := \{(x,u) \in D(\Sigma,\Sigma') : \text{there is a string } w \in S(x,u) \text{ such that } P(\Sigma,x,w) \bigcap \rho(\Sigma) = \varnothing\}. \tag{4-5}$$

For elements $(x,u) \in D_N(\Sigma,\Sigma')$, the machine Σ can reach the value $s'(x,u)$ without passing through a cycle. Here, the controller will simply feed to Σ the string w,

so that the next stable state of the combination Σ_c will match the next stable state of the model Σ'.

Finally, on the difference set

$$D_C(\Sigma,\Sigma') := D(\Sigma,\Sigma')\backslash D_N(\Sigma,\Sigma'), \tag{4-6}$$

the machine Σ cannot match the response of Σ' without passing through a cycle. In this case, the controller will have to generate an input string for Σ that will initially drive the machine into a cycle, and then take it out of the cycle. When Σ comes out of the cycle, its next state may be one of several possible states, as exit from a cycle may give rise to a critical race. The controller will then have to resolve the indeterminancy of the race and drive the machine to the next stable state determined by the model Σ'. The exact process by which this is accomplished is described in the following proof.

Proof 4-1. Assume first that condition (i) of Theorem 4-1 is valid, i.e., that there exists a controller C such that $\Sigma_{c|s} = \Sigma'$. Let s_c be the stable recursion function of Σ_c, and recall that s' is the stable recursion function of the model Σ'. The last equality implies that $s_c(x,u) = s'(x,u)$ for all pairs $(x,u) \in X\times A$. Now, since the controller C accesses Σ only through its input, there is a string $w(x,u) \in A^+$ such that $s_c(x,u) = s(x,w(x,u))$ where s is the stable recursion function of Σ. Consequently, $s(x,w(x,u)) = s'(x,u)$, and the state $s'(x,u)$ is stably reachable from the state x in the machine Σ. Thus, any state that is stably reachable from the state x in the machine Σ' must also be stably reachable from the state x in the machine Σ. In other words, $K(\Sigma) \geq K(\Sigma')$, and we conclude that part (i) implies part (ii).

Conversely, assume that (ii) is true, i.e., that $K(\Sigma) \geq K(\Sigma')$. We construct below a controller C that satisfies statement (i) of Theorem 4-1. Recalling the set $D(\Sigma,\Sigma')$ of Equation 4-2, note that if $D(\Sigma,\Sigma') = \varnothing$, then the stable transition function of Σ matches that of Σ', and no controller is needed. Next, consider the case $D(\Sigma,\Sigma') \neq \varnothing$. Define the set D by replacing in $D(\Sigma,\Sigma')$ all the pairs included in cycles by the corresponding generalized pairs, that is,

$$D_C := \{D_C(\Sigma,\Sigma') \setminus \rho(\Sigma)\} \bigcup \{(x^{n+i},u) : \rho_i \bigcap D(\Sigma,\Sigma') \neq \varnothing\} \tag{4-7}$$

The set of all input characters that pair with a state x in D is given by

$$D(x) := \{u \in A: (x,u) \in D\}. \tag{4-8}$$

Let $\pi_x : X^\rho \times A \rightarrow X^\rho : (x,u) \mapsto x$ be the standard projection. Then, $\pi_x D$ is the set of all generalized states $x \in X^\rho$ for which there is an input character $u \in A$ such that $(x,u) \in D$. Also, let $U(x)$ be the set of all input characters that form a stable combination with the generalized state $x \in X^\rho$. The set

$$V := \{x \times U(x): x \in \pi_x D\} \tag{4-9}$$

is then the set of all generalized stable combinations with states in $\pi_x D$.

Suppose that $D_N(\Sigma,\Sigma') \neq \varnothing$. Then, for each $(x,u) \in D_N(\Sigma,\Sigma')$, there exists an input string $w^0(x,u)$ such that $s(x,w^0(x,u)) = s'(x,u)$ and the path $P(\Sigma,x,w)$ does not contain any cycle states. Let $m(0;x,u) = |w^0(x,u)|$ be the length of this string, write it in terms of its characters as $w^0(x,u) = \overset{0}{w_0}(x,u)\overset{0}{w_1}(x,u) \ldots \overset{0}{w_{m(x,u)-1}}(x,u)$. Then, $w^0(x,u)$ takes the machine through a succession of stable states, which we denote as follows.

$$x^{0,1}(x,u) := s(x,\overset{0}{w_0}(x,u)),$$

$$x^{0,2}(x,u) := s(x^{0,1}(x,u),w_1^0(x,u)),\ \ldots,$$
$$x^{0,m(x,u)-1}(x,u) := s(x^{0,m(x,u)-2}(x,u),w_{m(x,u)-2}^0(x,u)),$$

and, by selection of the string $w^0(x,u)$,

$$s(x^{0,m(x,u)-1}(x,u),w_{m(x,u)-1}^0(x,u)) = s'(x,u). \tag{4-10}$$

Thus, to guarantee fundamental mode operation, we see that the controller must have $m(0;x,u)$ states associated with the pair (x,u). Now, let $(z^1,u^1), ..., (z^{n(N)},u^{n(N)})$ be the elements of $D_N(\Sigma,\Sigma')$. Then, for each pair $(z^i,u^i) \in D_N(\Sigma,\Sigma')$, the controller needs $m(0;z^i,u^i)$ states to facilitate fundamental mode operation, say the states $\xi_1(z^i,u^i), \ldots, \xi_{m(0;z^i,u^i)}(z^i,u^i)$. This leads to the set of controller states

$$\Xi_N := \{\xi_1(z^1,u^1), \ldots, \xi_{m(0;z^1,u^1)}(z^1,u^1), \ldots, \xi_1(z^{n(N)},u^{n(N)}), \ldots, \xi_{m(0;z^{n(N)},u^{n(N)})}(z^{n(N)},u^{n(N)})\}. \tag{4-11}$$

In case $D_N(\Sigma,\Sigma') = \varnothing$, set $\Xi_N := \varnothing$.

We turn now to the set D_C of Equation 4-7, assuming that it is not empty. Let $n(C)$ be the number of elements of D_C. Recall that, for each pair of D_C, the machine Σ must pass through at least one cycle in the process of matching the response of the model Σ'; one of these cycles may be the initial step.

Consider a pair $(x,u) \in D_C$, let $T(\Sigma_{|s})$ be the matrix of stable transitions of Σ. Recall that the state set of Σ is $X = \{x^1, ..., x^n\}$ and the generalized state set of Σ is $X^\rho = \{x^1, ..., x^{n+t}\}$. In the case x is a cycle state, say $x = x^\gamma$, where $\gamma \in \{n+1, ..., n+t\}$, let $\varphi(x^\gamma) := \{x^{\alpha(1)}, ...,x^{\alpha(p)}\} \subset X$. Setting $x^\beta := s'(x^{\alpha(1)},u) \in X$, it follows by Proposition 3-19 that $s'(x^{\alpha(i)},u) = x^\beta$ for all $i = 1, ..., p$. The fact that $x^\beta := s'(x^{\alpha(i)},u)$ implies that $K_{\beta\alpha(i)}(\Sigma') = 1$, $i = 1, ..., p$. By Equation 4-1, we then have that $K_{\beta\gamma}(\Sigma') = 1$. The

inequality $K(\Sigma) \geq K(\Sigma')$ implies then that the entry $T_{\beta\gamma}(\Sigma_{|s})$ is not empty. Furthermore, letting $w(x,u)$ be an element of $T_{\beta\gamma}(\Sigma_{|s})$, it follows by Proposition 3-18 that $s'(x,u) = s(x,w(x,u))$. Consider now the path $P(\Sigma_{|s},x,w(x,u))$. Recalling that $(x,u) \in D_C$, the following hold. If x is not a cycle state, then the path $P(\Sigma_{|s},x,w(x,u))$ must include a cycle. If x is a cycle state, then $P(\Sigma_{|s},x,w(x,u))$ may or may not include an additional cycle.

The string $w(x,u)$ takes the machine through one possible outcome of each cycle encountered along the path $P(\Sigma_{|s},x,w(x,u))$. Depending on the outcome of the critical races that occur as the machine Σ passes through the cycles along the path $P(\Sigma_{|s},x,w(x,u))$, different strings may need to be used to take the machine to the state $s'(x,u)$. Let $w_0(x,u)$ be the initial segment of the string $w(x,u)$ that takes the machine Σ to the state following the first cycle state along the path $P(\Sigma_{|s},x,v(x,u))$. By construction, the shortest length of $w^0(x,u)$ is one character, and it is obtained when the state x is itself a cycle state. The controller C initially feeds this string to the machine Σ character by character, to achieve semi-fundamental mode operation. The last character of $w^0(x,u)$ leads Σ into a critical race, as it exits from the corresponding cycle. The inequality $K(\Sigma) \geq K(\Sigma')$ implies that there is an input string that leads Σ from each outcome of this race to the next cycle along its way to the state $s'(x,u)$. This string will then similarly be generated by the controller and applied to the machine Σ. This process continues through each cycle encountered along the way, and is formally described next.

Writing the initial substring in terms of its individual characters, set $w^0(x,u) = w_0^0(x,u)w_1^0(x,u) \ldots w_{m(0;x,u)-1}^0(x,u)$; here, $m(0;x,u) = |w^0(x,u)|$ is the length of the string.

This string take the machine Σ through the following sequence of generalized stable states.

$$
\begin{aligned}
&x^{0,1}(x,u) := s(x,w^{0}_{0}(x,u)),\\
&x^{0,2}(x,u) := s(x^{0,1}(x,u),w^{0}_{1}(x,u)),\ \ldots,\\
&x^{0,m(0;x,u)-1}(x,u) := s(x^{0,m(0;x,u)-2}(x,u),w^{0}_{m(0;x,u)-2}(x,u)),\\
&X^{(1)}(x,u) := s(x^{0,m(0;x,u)-1}(x,u),w^{0}_{m(0;x,u)-1}(x,u)),
\end{aligned}
\tag{4-12}
$$

where $X^{(1)}(x,u) = \{x(1,1), \ldots, x(1,r_1)\} \subset X$ is the set of all states of Σ that form possible outcomes of the critical race. Since the string $w^0(x,u)$ takes Σ up to one step past a cycle state to the outcome of the corresponding critical race, it follows that $x^{0,m(0;x,u)-1}(x,u)$ is a cycle state. Note that if the initial state x is a cycle state, then $m(0;x,u) = 1$, namely, the substring $w^0(x,u)$ has just a single character.

The next substring of characters in $w(x,u)$ takes the machine from the present outcome of the cycle to the outcome of the next cycle (or to the final destination, if no other cycle is encountered). Denote this substring by $w_1(x,u,x(1,i_1))$. It takes the machine from the state $x(1,i_1)$ to the outcome of the next cycle; the process is similar to the process described in Equation 4-12, and $w^1(x,u,x(1,i_1))$ has at least one character. If a cycle leading to a critical race is reached at the end of the substring $w^1(x,u,x(1,i_1))$, then, letting $r_2(i_1)$ be the number of its outcomes, we denote the set of outcomes by $X(2,i_1)(x,u)$ and let its members be $\{x(2,i_1,1), \ldots, x(2,i_1,r_2(i_1))\} \subset X$. Let $m(1;x,u,x(1,i_1)) = |w^1(x,u,x(1,i_1))|$ be the length of the string, and let $w^1_0(x,u,x(1,i_1))$, $w^1_1(x,u,x(1,i_1))$, $\ldots$, $w^1_{m(1;x,u,x(1,i_1))-1}(x,u,x(1,i_1))$ be its characters.

The next such substring depends on the outcome of both the races encountered, and is given by $w^2(x,u,x^{(2,i_1,i_2)})$. The number of cycles encountered may vary, depending on

the outcome of the critical races through which the machine has passed. Let q be the number of cycles encountered along our path to the target $s'(x,u)$. The outcomes of the q critical races caused by the cycles are then denoted by $x^{(1,i_1)}, x^{(2,i_1,i_2)}, \ldots, x^{(q,i_1,\ldots,i_q)}$. In order to simplify the notation, define $x(k,i_{(\le k)}) = x(k,i_1, \ldots, i_k)$, for some k. The string of input characters that takes the machine from (x,u) to $s'(x,u)$ consists then of the following segments

$$w(x,u) := w^0(x,u)\, w^1(x,u,x(1,i_{(\le 1)})) \ldots w^q(x,u,x(q,i_{(\le q)})), \tag{4-13}$$

where each one of the segments takes the machine from the outcome of one cycle to the outcome of the next cycle along the path. The length of the segment $w^j(x,u,x(j,i_{(\le j)}))$ is $m(j;x,u,x(j,i_{(\le j)})) \ge 1$. Each segment can then be written in terms of its individual characters in the form

$$w^j(x,u,x(j,i_{(<j)})) := w_0^j(x,u,x(j,i_{(\le j)}))\, w_1^j(x,u,x(j,i_{(\le j)})) \ldots w_{m(j;x,u,x(j,i_{(\le j)}))-1}^j(x,u,x(j,i_{(\le j)})), \tag{4-14}$$

where the possible values for each variable are $i_1 = 1, \ldots, r_1$, $i_2 = 1, \ldots, r_2(i_1), \ldots, i_j = 1, \ldots, r_j(i_1,\ldots,i_{j-1})$, and $j = 1, \ldots, q$. Using the same notation introduced earlier, let $r_j(i_{(\le j-1)}) = r_j(i_1,\ldots,i_{j-1})$ for some j.

Each of the substrings $\{w^j(x,u,x(j,i_{(\le j)}))\}_{j=1}^{q-1}$ drives the machine through a string of the stable transitions similar to Equation 4-12. The last segment $w^q(x,u,x(q,i_{(\le q)}))$ takes the machine to the destination $s'(x,u)$, i.e.,

$$s(x^{q,m(q;x,u,x(q,i_{(\le q)}))-1}(x,u,x(q,i_{(\le q)})), w_{m(q;x,u,x(q,i_{(\le q)}))-1}^q(x,u,x(q,i_{(\le q)}))) = s'(x,u).$$

We are now ready to introduce the operation of the controller C. The controller generates each segment of the string $w(x,u)$ in turn. To guaranty semi-fundamental

mode operation, the controller must generate the string segments $w^0(x,u)$, ..., $w^q(x,u,x(q,i_{(<q)}))$ one character at a time, waiting at each character until Σ has reached its next generalized state. For this purpose, the controller operation is defined as follows. Let ϕ be the transition function of the controller and let η be its output function.

Assume that the controller is in the initial state ξ_0 and that the machine Σ is in a generalized stable combination (x,v), when the input character changes to u. Upon detecting the combination $(x,u) \in V$, the controller changes to the state ξ_1. This entails the following values of the transition function.

$$\phi(\xi_0,(z,t)) := \xi_0 \text{ for all } (z,t) \in X\times A\backslash V,$$
$$\phi(\xi_0,(x,u)) := \xi_1 \text{ for all } (x,u) \in V. \qquad (4\text{-}15)$$

Referring to (a11), note that, while in the state ξ_0, the controller is transparent; it applies to the machine Σ it's own external input character. This entails the following value of the output function.

$$\eta(\xi_0,(z,t)) := t \text{ for all } (z,t) \in X_\rho\times A. \qquad (4\text{-}16)$$

For semi-fundamental mode operation, the machine Σ cannot change its generalized state during a state change of the controller C. To this end, recall that $U(x)$ is the set of all characters that form generalized stable combinations with the state x. Select a character $v' \in U(x)$ and set

$$\eta(\xi_1,(x,t)) := v' \text{ for all } (x,t) \in X_\rho\times A. \qquad (4\text{-}17)$$

The controller starts now to generate the substring $w^0(x,u)$, feeding it to the machine Σ in order to make its response match the response of the model Σ'. The substring $w^0(x,u)$ is generated one character at a time, waiting after each character for the machine Σ to reach its next generalized stable combination. This process requires

the controller states $\xi^0_0(x,u), \ldots, \xi^0_{m(0;x,u)-1}(x,u)$, and the controller's recursion function is defined as follows.

$$\begin{aligned}
&\phi(\xi_1,(x,u)) := \xi^0_0(x,u), \text{ for } (x,u) \in D,\\
&\phi(\xi_1,(x,t)) := \xi_1, \text{ for all } t \in U(x)\backslash D(x),\\
&\phi(\xi^0_j(x,u),(x^{0,j}(x,u),u)) := \xi^0_{j+1}(x,u),\\
&\phi(\xi^0_j(x,u),(z,t)) := \xi^0_j(x,u), \text{ for all } (z,t) \in X_\rho \times A,
\end{aligned} \tag{4-18}$$

where $0 \le j \le m(0;x,u)-2$. We can now generate the first $m(0;x,u)-1$ characters of the substring $w^0(x,u)$ by defining the controller output function as follows.

$$\eta(\xi^0_j(x,u),(z,t)) := w^0_j(x,u), \text{ for all } (z,t) \in X_\rho \times A, \tag{4-19}$$

where $0 \le j \le m(0;x,u)-1$. The final character of $w^0(x,u)$ is applied while the machine Σ is in the cycle state $x^{0,m(0;x,u)-1}(x,u)$, and creates a critical race with the outcome $x(1,i_1)$, as discussed earlier.

Recall that $w(x,u)$ passes through q cycles. The controller continues to generate all the substrings $w^j(x,u,x(j,i_{(\le j)}))$, $j = 1, \ldots, q$ consecutively. Let the controller states associated with the substring $w^j(x,u,x(j,i_{(\le j)}))$ be $\xi^j_0(x,u), \ldots, \xi^j_{m(j;x,u,x(j,i_{(\le j)}))-1}(x,u)$. The transition function is then given by

$$\begin{aligned}
&\phi(\xi^j_h(x,u),(x^{j,h}(x,u),u)) := \xi^j_{h+1}(x,u),\\
&\phi(\xi^j_h(x,u),(z,t)) := \xi^j_h(x,u), \text{ for all } (z,t) \ne (x^{j,h}(x,u),u),
\end{aligned} \tag{4-20}$$

where $h = 0, \ldots, m(j;x,u,x(j,i_{(\le j)}))-2$, and $j = 1, \ldots, q$. The corresponding output values are generated by the controller's output function as follows.

$$\eta(\xi^j_h(x,u),(z,t)) := w^j_h(x,u,x(j,i_{(\le j)})), (z,t) \in X_\rho \times A, \tag{4-21}$$

where $h = 0, \ldots, m(j;x,u,x(j,i_{(\le j)}))-1$, and $j = 1, \ldots, q$.

The transition from the end of one substring to the first character of the next substring of $w(x,u)$ is accomplished by defining the controller's transition function as follows.

$$\phi(\xi^{j}_{m(j;x,u,x(j,i_{(\le j)}))-1}(x,u),(x^{j,h}(x,u),u)) := \xi^{j+1}_{0}(x,u), \tag{4-22}$$

$j = 0, \ldots, q-1$.

At the end of the string $w(x,u)$, the machine Σ reaches the state $s'(x,u)$, required for matching the model Σ'. The last character of $w(x,u)$ is, of course, the last character of $w^{q}(x,u,x(q,i_{(\le q)}))$. After this character is applied, the machine Σ must remain in a stable combination with the state $s'(x,u)$ until the external input changes. To implement this requirement, the transition function of the controller is defined by

$$\phi(\xi^{q}_{m(q;x,u,x(q,i_{(\le q)}))-1}(x,u),(z,t)) := \xi_1, \ (z,t) \neq (s'(x,u),u). \tag{4-23}$$

This completes the operation of the controller for the transition from x to $s'(x,u)$. Summarizing, the set of all controller states associated with the generation of the string $w(x,u)$ is given by

$$\Xi_C(w(x,u)) = \{\xi^{0}_{0}(x,u), \ldots, \xi^{0}_{m(0;x,u)-1}(x,u), \ldots, \xi^{q}_{0}(x,u), \ldots, \xi^{q}_{m(q;x,u,x(q,i_{(\le q)}))-1}(x,u)\}. \tag{4-24}$$

Note when $(x,u) \in D_N(\Sigma,\Sigma')$, then q is taken as zero in the above construction, and $w(x,u)$ consists of only one substring which completes the required transition.

To complete the implementation of the controller C, the above construction has to be applied to each input pair $(x,u) \in D_C$. To describe the state set of the controller, let the elements of D_C be given by $\{(z^1,u^1), \ldots, (z^{n(C)},u^{n(C)})\}$ and let $w(z^k,u^k)$ be the string that takes Σ from (x^k,u^k) to the state $s'(x^k,u^k)$ required to match the model for a

specific sequence of cycle outcomes. Then, in analogy with Equation 4-24, the controller states associated with the generation of $w(x^k,u^k)$, for a specific combination of the outcome of the cycles encountered, are

$$\Xi_C(w(x^k,u^k)) = \{\xi_0^0(x^k,u^k), \ldots, \xi^0_{m(0;x^k,u^k)-1}(x^k,u^k), \ldots, \xi_0^q(x^k,u^k), \ldots, \xi^q_{m(q;x^k,u^k,x(q,i_{(\le q)})-1)}(x^k,u^k)\}. \quad (4\text{-}25)$$

The set of all states for all combinations of outcomes of cycles are then given by

$$\Xi_C(x^k,u^k) = \bigcup_{\text{all possible } w(x^k,u^k)} \Xi_C(w(x^k,u^k)). \quad (4\text{-}26)$$

Thus, the set of controller states for all the elements of D_C are

$$\Xi_C = \bigcup_{k = 1, \ldots, n(C)} \Xi_C(x^k,u^k) \quad (4\text{-}27)$$

Of course, if $D_C = \varnothing$, then $\Xi_C(x^k,u^k) = \varnothing$ and $\Xi_C = \varnothing$. Then the complete state set of the controller is given by

$$\Xi = \{\xi_0, \xi_1\} \cup \Xi_N \cup \Xi_C. \quad (4\text{-}28)$$

By construction, the closed loop system operates in semi-fundamental mode and is well posed. This completes the construction of the controller C. ♦

Note that the states of the controller are not optimized. This can be done by the various state reduction techniques, e.g., KOHAVI [1970]. Further state reductions can be done by careful choice of the corrective string.

CHAPTER 5
EXAMPLE

This chapter demonstrates the construction of a controller C that fulfills the requirements of Theorem 4-1. Consider the input/state machine Σ having the input set $A = \{a, b, c, d\}$ and the state set $X = \{x^1, x^2, x^3, x^4\}$. Consider the recursion function of the machine f to be defined by Table 5-1 of transitions.

Table 5-1. Transition table for the machine Σ.

	a	b	c	d
x^1	x^1	x^3	x^1	x^1
x^2	x^3	x^2	x^4	x^3
x^3	x^4	x^4	x^4	x^3
x^4	x^2	x^4	x^3	x^4

Applying Algorithm 3-8 to the above machine, the cycles in the machine can be found. We see that the cycles are $\rho_1 = \{a;x^2, x^3, x^4\}$ and $\rho_2 = \{c;x^3, x^4\}$. Now, the cycle states corresponding to the cycles ρ_1 and ρ_2 are x^5 and x^6. The generalized stable state machine $\Sigma_{|s}$ now has the state set $X^\rho = \{x^1, x^2, x^3, x^4, x^5, x^6\}$, as defined in Equation 3-8, and the same input set A. The stable recursion function s of the generalized stable state machine $\Sigma_{|s}$ is defined in the Table 5-2 of transitions. The generalized stable recursion function for the generalized stable state machine is defined in Equation 3-11. Note that when an input character is applied to a cycle state, it can result in unpredictable outcomes. This is represented as the set of all possible output states.

Table 5-2. Stable transition table for the generalized stable state machine $\Sigma_{|s}$

	a	b	c	d
x^1	x^1	x^3	x^1	x^1
x^2	x^5	x^2	x^6	x^3
x^3	x^5	x^4	x^6	x^3
x^4	x^5	x^4	x^6	x^4
x^5	x^5	$\{x^2,x^4\}$	$\{x^4,x^6\}$	$\{x^3,x^4\}$
x^6	x^5	x^4	x^6	$\{x^3,x^4\}$

The machine $\Sigma_{|s}$ is now an asynchronous machine with multiple race outcomes. We will now follow the procedure in Chapter 4 to calculate the skeleton matrix of this machine. As defined in Equation 3-12, the matrix of one-step generalized stable transitions $R(\Sigma_{|s})$ is shown in Table 5-3.

Table 5-3. Matrix of one-step generalized stable transitions $R(\Sigma_{|s})$

	x^1	x^2	x^3	x^4	x^5	x^6
x^1	a,c,d	N	N	N	N	N
x^2	N	b	N	N	b	N
x^3	N	d	d	N	d	d
x^4	b	N	b	b,d	b,d	b,d
x^5	N	a	a	a	a	a
x^6	N	c	c	c	c	c

Using Equation 3-17 and Equation 3-18, we construct the matrices $R^2(\Sigma_{|s})$, $R^3(\Sigma_{|s})$, $R^4(\Sigma_{|s})$, and $R^5(\Sigma_{|s})$, and then the matrix $R^{(5)}(\Sigma_{|s})$. Then, applying Definition 3-14, the matrix of stable transitions $T(\Sigma_{|s})$ is constructed. Finally, using Definition 3-17, the skeleton matrix is derived. The skeleton matrix $K(\Sigma)$ calculated from $R^{(5)}(\Sigma)$ is given by

$$K(\Sigma) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{5-1}$$

Now to solve the model matching problem, we need the given stable state machine Σ' as our model. The stable recursion function for Σ' is given in Table 5-4 of transitions.

Table 5-4. Stable transition function of the given model Σ'.

	a	b	c	d
x^1	x^1	x^3	x^1	x^1
x^2	x^4	x^2	x^4	x^3
x^3	x^4	x^4	x^4	x^3
x^4	x^4	x^4	x^4	x^4

Now, the skeleton matrix for the model follows the same procedure as in the generalized stable state machine, except for the absence of the cycle states. This results in a square matrix, given by

$$K^0(\Sigma') = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{5-2}$$

Following Equation 4-1, the augmented skeleton matrix $K(\Sigma')$ is then constructed as follows.

$$K(\Sigma') = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{5-3}$$

The next step in the construction of the controller is to see if the necessary and sufficient conditions are satisfied. We find that $K(\Sigma) \geq K(\Sigma')$, thus satisfying Theorem 4-1. To construct the controller, we need the following matrices, as explained in the proof of Theorem 4-1.

$$\begin{aligned} &D(\Sigma,\Sigma') := \{(x^2,a),(x^3,a),(x^4,a),(x^2,c),(x^3,c),(x^4,c)\}, \\ &D := \{(x^5,a),(x^6,c),(x^2,c)\}, \\ &\pi_x D := \{x^2,x^3,x^4\}, \\ &D(x) := \{a,c\}, \\ &U(x^2) := \{b\}, \\ &U(x^5) := \{a\}, \\ &U(x^6) := \{c\}, \\ &D_N(\Sigma,\Sigma') := \varnothing, \\ &D_C := \{(x^2,c),(x^5,a),(x^6,c)\}. \end{aligned} \tag{5-4}$$

For $(x^2,c) \in D_C(\Sigma,\Sigma')$, $w_0(x^2,c) = d$, $w_1(x^2,c,x^3) = b$. There are two possible strings, d and db; $r(x^2,c) = 2$. The state transitions are:

$$\begin{aligned} &\phi(\xi_0,(z,t)) := \xi_0 \text{ for all } (z,t) \in X\times A\backslash V, \\ &\phi(\xi_0,(x^2,c)) := \xi_1 \text{ for all } (x^2,c) \in V, \\ &\phi(\xi_1,(x^2,c)) := \xi_0^0(x2,c), \text{ for } (x^2,c) \in D, \\ &\phi(\xi_1,(x^2,t)) := \xi_1, \text{ for all } t \in U(x^2)\backslash D(x^2), \\ &\phi(\xi_1,(z,t)) := \xi_0, \text{ for all pairs } (z,t) \in X\rho\times A\backslash(V\cup D), \\ &\phi(\xi_0^0(x2,c),(x^{0,1}(x^2,c),d)) := \xi_0^1(x^2,c), \\ &\phi(\xi_0^0(x2,c),(z,t)) := \xi_0^0(x2,c), \text{ for all } (z,t) \neq (x^{0,1}(x^2,c),d), \\ &\phi(\xi_0^1(x2,c),(z,t)) := \xi_1, (z,t) \neq (s'(x^2,c),u). \end{aligned} \tag{5-5}$$

The output functions are

$\eta(\xi_0,(z,t)) := t$ for all $(z,t) \in X_\rho \times A$,
$\eta(\xi_1,(x^2,t)) := b$ for all $(x^2,t) \in X_\rho \times A$,
$\eta(\overset{0}{\xi_0}(x^2,c),(z,t)) := \overset{0}{w_0}(x^2,c) = d$, for all $(z,t) \in X_\rho \times A$,
$\eta(\overset{1}{\xi_0}(x^2,c),(z,t)) := \overset{1}{w_0}(x^2,c,x^3) = b$, $(z,t) \in X_\rho \times A$. (5-6)

Now, consider the next element $(x^5,a) \in D_C(\Sigma,\Sigma')$. For this cycle state, $w_0(x^5,a) = d$, $w_1(x^5,a,x^3) = b$; as before, $r(x^5,a) = 2$. The state transitions and output functions are given by

$\phi(\xi_0,(z,t)) := \xi_0$ for all $(z,t) \in X \times A \backslash V$,
$\phi(\xi_0,(x^5,a)) := \xi_1$ for all $(x^5,a) \in V$,
$\phi(\xi_1,(x^5,a)) := \overset{0}{\xi_0}(x^5,a)$, for $(x^5,a) \in D$,
$\phi(\xi_1,(x^5,t)) := \xi_1$, for all $t \in U(x^5) \backslash D(x^5)$,
$\phi(\xi_1,(z,t)) := \xi_0$, for all pairs $(z,t) \in X_\rho \times A \backslash (V \cup D)$,
$\phi(\overset{0}{\xi_0}(x^5,a),(x^{0,1}(x^5,a),d)) := \overset{1}{\xi_0}(x^5,a)$,
$\phi(\overset{0}{\xi_0}(x^5,a),(z,t)) := \overset{0}{\xi_0}(x^5,a)$, for all $(z,t) \neq (x^{0,1}(x^5,a),d)$,
$\phi(\overset{1}{\xi_0}(x^5,a),(z,t)) := \xi_1$, $(z,t) \neq (s'(x^5,a),b)$,
$\eta(\xi_0,(z,t)) := t$ for all $(z,t) \in X_\rho \times A$,
$\eta(\xi_1,(x^5,t)) := a$ for all $(x^5,t) \in X_\rho \times A$,
$\eta(\overset{0}{\xi_0}(x^5,a),(z,t)) := \overset{0}{w_0}(x^5,a) = d$, for all $(z,t) \in X_\rho \times A$,
$\eta(\overset{1}{\xi_0}(x^5,a),(z,t)) := \overset{1}{w_0}(x^5,a,x^3) = b$, $(z,t) \in X_\rho \times A$. (5-7)

Similarly, the transition and output functions for the third element $(x^6,a) \in D_C(\Sigma,\Sigma')$ are defined below. Here, $w_0(x^6,c) = b$; $r(x^6,a) = 1$.

$\phi(\xi_0,(z,t)) := \xi_0$ for all $(z,t) \in X \times A \backslash V$,
$\phi(\xi_0,(x^6,c)) := \xi_1$ for all $(x^6,c) \in V$,
$\phi(\xi_1,(x^6,c)) := \overset{0}{\xi_0}(x^6,c)$, for $(x^6,c) \in D$,
$\phi(\xi_1,(x^6,t)) := \xi_1$, for all $t \in U(x^6) \backslash D(x^6)$,
$\phi(\xi_1,(z,t)) := \xi_0$, for all pairs $(z,t) \in X_\rho \times A \backslash (V \cup D)$,
$\phi(\overset{0}{\xi_0}(x^6,c),(z,t)) := \xi_1$, $(z,t) \neq (s'(x^6,c),b)$,
$\eta(\xi_0,(z,t)) := t$ for all $(z,t) \in X_\rho \times A$,
$\eta(\xi_1,(x^6,t)) := c$ for all $(x^6,t) \in X_\rho \times A$,

$$\eta(\overset{0}{\xi_0}(x^6,c),(z,t)) := \overset{0}{w_0}(x^6,c) = b, \text{ for all } (z,t) \in X_\rho \times A. \qquad (5\text{-}8)$$

This concludes the construction of the feedback controller C such that the closed loop machine $\Sigma_{c|s} = \Sigma'$. The equations (5-6), (5-7) and (5-8) give the state and transition functions of the controller C.

CHAPTER 6
SUMMARY AND FUTURE WORK

The present work shows the use of feedback controllers to correct one of the common hazards in the behavior of asynchronous sequential machines. Specifically, the designed controller not only eliminates the effects of infinite cycles in such a machine, but it can also drive the defective machine to match the behavior of a desired model. This approach opens an interesting arena where many open problems remain to be analyzed.

In Chapter 5 of this dissertation, solutions have been presented to the model matching problem of input-state machines involved in infinite cycles. The necessary and sufficient conditions for the existence of a controller have been stated, and an algorithm for its construction has been included, whenever a controller exists.

The control of asynchronous machines is an area of research that is still ripe with possibilities. A straight forward extension of this dissertation is to the construction of a comprehensive state feedback controller that solves the the model matching problem for a machine with multiple hazards, including both races and cycles. The comprehensive controller has elements of both the race problem investigated by Murphy 1996, Murphy, Geng, and Hammer 2002, and Murphy, Geng, and Hammer 2003, as well as the controller for remedying cycles in an asynchronous machine, described in this dissertation.

My future research plans include work on extending the results obtained in this dissertation to the comprehensive state feedback controller. The next logical step lies in

formulating a more compact representation of the controller. A close examination of the controller structure described in Chapter 3 and Chapter 4 shows that it consists of a number of interconnected similar units. This observation leads us to believe that the controller can be assembled in a modular fashion from a collection of standard units. Formalizing this observation will lead to a simple methodology of controller design, whereby the controller is constructed simply by interconnecting a few standard blocks.

Further, investigating the possibility of a comprehensive output feedback controller for a machine with multiple races and cycles is also a future research interest. The present proposal develops a controller that eliminates the negative effects of infinite cycles in asynchronous machines under the assumption that the state of the machine is available for feedback purposes. The problem of overcoming the effects of infinite cycles becomes substantially more complex if the state of the machine is not available, and only an output signal is available. The solution of this problem requires the development of concepts related to the observability of asynchronous machines, to find conditions under which an infinite cycle in state space can be detected from an observation of the output signal.

The application of the obtained results to parallel and distributed systems is also an area of interest. Asynchronous sequential systems can be used to model parallel and distributed systems due to their event driven nature. It has been shown by Plateau and Atif 1991 that complex parallel networks can be modeled using stochastic automata networks. Recent work has succeeded in showing that a distributed system can be represented by cellular automata (Zambonelli et al 2003). Feedback control holds

promise for improving the performance and reliability of such parallel and distributed systems, and it is possible to apply the principles developed in this work.

LIST OF REFERENCES

Alpan, G., and Jafari, M.A., 2002, "Synthesis of a closed-loop combined plant and controller model," IEEE Transactions on Systems, Man and Cybernetics, Vol. 32, No. 2, pp.163-175.

Arbib, M., 1969, "Theories of abstract automata," Prentice Hall Inc., Englewood Cliffs, NJ.

Armstrong, D., Friedman A., and Menon P., 1968, "Realization of asynchronous sequential circuits without inserted delay elements," IEEE Transactions on Computers, C-17, No. 2.

Bredeson, J., and Hulina, P., 1971, "Generation of a clock pulse for asynchronous sequential machines to eliminate critical races," IEEE Transactions on Computers, C-20, pp. 225-226.

Bruschi, D., Pighizzini, G., and Sabadini, N., 1994, "On the existence of minimum asynchronous automata and on the equivalence problem for unambiguous regular trace languages," Information and Computation, Vol. 108, No. 2, pp. 262-285.

Brzozowski, J.A., and Seger, C., 1987, "A characterization of ternary simulation of gate networks," IEEE Transactions on Computers, C-36, No. 11, pp. 1318-1327.

Brzozowski, J.A., and Seger, C-J., 1989, "A unified framework for race analysis of asynchronous networks," Journal of the Association for Computing Machinery, Vol. 36, No. 1, pp. 20-45.

Brzozowski, J.A., and Yeoli, M., 1979, "On a ternary model of gate networks," IEEE Transactions on Computers, C-28, pp.178-183.

Chiang, J., and Radhakrishnan, D., 1990, "Hazard-free design of mixed operating mode asynchronous sequential circuits," International Journal of Electronics, Vol. 68, No. 1, pp. 23-37.

Chu, T., 1994, "Synthesis of hazard-free control circuits from asynchronous finite state machine specifications," Journal of VLSI Signal Processing, Vol. 7, No. 1-2, pp. 61-84.

Chu, T., Mani, N., and Leung, C., 1993, An efficient critical race-free state assignment technique for asynchronous finite state machines," Proceedings of the 30th ACM/IEEE Design Automation Conference, Dallas, TX, pp. 2-5.

Cole, R., and Zajicek, O., 1990, "The expected advantage of asynchrony," Proceedings of the Second Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 85-94.

Datta, P.K., Bandyopadhyay, S.K., and Choudhury, A.K., 1988, "A graph theoretic approach for state assignment of asynchronous sequential machines," International Jounal of Electronics, Vol. 65, No. 6, pp. 1067-1075.

Davis, A., Coates, B., and Stevens, K., 1993a, "Automatic synthesis of fast compact asynchronous control circuits," S. Furber and M. Edwards, editors, Asynchronous Design Methodologies, Volume A-28 of IFIP Transactions, Elsevier Science Publishers, pp. 193-207.

Davis, A., Coates, B., and Stevens, K., 1993b, "The post office experience: designing a large asynchronous chip," Proceeding of the 26th Hawaii International Conference on System Sciences, Vol. 1, pp. 409-418.

Dibenedetto, M.D., Saldanha, A., and Sangiovanni-Vincentelli, A., 1994, "Model matching for finite state machines," Proceedings of the IEEE Conference on Decision and Control, Vol. 3, pp. 3117-3124.

Dibenedetto, M.D., Saldanha, A., and Sangiovanni-Vincentelli, A., 1995a, "Strong model matching for finite state machines with nondeterministic reference model," Proceedings of the IEEE conference on Decision and Control, Vol. 1, pp. 422-426.

Dibenedetto, M.D., Saldanha, A., and Sangiovanni-Vincentelli, A., 1995b, "Strong model matching for finite state machines," Proceedings of European Control Conference, Vol. 3, pp. 2027-2034.

Dibenedetto, M.D., Sangiovanni-Vincentelli, A., and Villa, T., 2001, "Model matching for finite-state machines," IEEE Transactions on Automatic Control, Vol. 26, No. 11, pp. 1726-1743.

Eichelberger, E.B., 1965, "Hazard detection in combinational and sequential switching circuits," IBM Journal of Research and Development, Vol. 9, No. 2, pp. 90-99.

Eilenberg, S., 1974, "Automata, Languages and Machines," Volume A, Academic Press, New York.

Evans, J.B., 1988, "Structures of discrete event simulation," Halsted Press, New York.

Fisher, P., and Wu, S., September 1993, "Race-free state assignments for synthesizing large-scale asynchronous sequential logic circuits," IEEE Transactions of Computers, Vol. 42, No. 9, pp.1025-1034.

Fujita, M., 1993, "Methods for automatic design error correction in sequential circuits," Proceedings of the European Conference on Design Automation with the European Event in ASIC Design, pp. 76-80.

Furber, S.B., 1993, “Breaking step - The return of asynchronous logic,” IEEE Review, pp. 159-162.

Geng, X. J., 2003, “Model matching for asynchronous sequential machines,” Ph.D. dissertation, Department of Electrical and Computer Engineering, University of Florida, Gainesville, Florida.

Geng, X. J., and Hammer, J., 2003, “Input/output control of asynchronous sequential machines”, submitted for publication.

Geng, X. J., and Hammer, J., 2004a, “Asynchronous sequential machines: input/output control”, Proceedings of the 12th Mediterranean Conference on Control and Automation, Kusadasi, Turkey, June 2004.

Geng, X. J., and Hammer, J., 2004b, “Output Feedback Control Asynchronous Sequential Machines”, Proceedings of the 8th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2004) to be held in Orlando, Florida.(to appear).

Hammer, J., 1994, “On some control problems in molecular biology,” Proceedings of the IEEE Conference on Decision and Control, Vol. 4, pp. 4098-4103.

Hammer, J., 1995, “On the modeling and control of biological signaling chains,” Proceedings of the IEEE Conference on Decision and Control, Vol. 4, pp. 3747-3752.

Hammer J., 1996a, “On the control of incompletely described sequential machines,” International Journal of Control, Vol. 63, No. 6, pp. 1005-1028.

Hammer J., 1996b, “On the corrective control of sequential machines,” International Journal of Control, Vol. 65, No. 6, pp. 249-276.

Hauck, S., 1995, “Asynchronous design methodologies: an overview,” Proceedings of the IEEE, Vol. 83, No. 1, pp. 69-93.

Hazeltine, B., 1965, “Encoding of asynchronous sequential circuits,” IEEE Transactions on Electronic Computers, EX-14, pp.727-729.

Higham, L., and Schenk, E., 1992, “The parallel asynchronous recursion model,” Proceedings of the IEEE Symposium on Parallel and Distributed Processing, pp. 310-316.

Hlavicka, J., 1970, “Essential hazard correction without the use of delay elements,” IEEE Transactions on Computers, C-19, No. 3, pp. 232-238.

Holcombe, W.M.L., 1982, “Algebraic automata theory,” Cambridge University Press, New York.

Hubbard, P., and Caines, P.E., 2002, "Dynamical consistency in hierarchical supervisory control," IEEE Transactions on Automatic Control, Vol. 47, No. 1, pp. 37-52.

Huffmann D.A., 1954a, "The synthesis of sequential switching circuits," J. Franklin Inst., Vol. 257, pp. 161-190.

Huffmann D.A., 1954b, "The synthesis of sequential switching circuits," J. Franklin Inst., Vol. 257, pp. 275-303.

Huffmann, D.A., 1957, "The design and use of hazard-free switching networks," Journal of the Association of Computing Machinery, Vol. 4, No. 1, pp. 47-62.

Ingerson, T.E., and Buvel, R.L., 1984, "Structure in Asynchronous Cellular Automata," Physica D, Vol. 10, pp. 59-68.

Kailath, T., 1980, "Linear systems," Prentice Hall Inc., Englewood Cliffs, New Jersey, 1979, pp. 268-272.

Kalman, R.E., Falb, P.L., and Arbib, M.A., 1969, "Topics in Mathematical Systems Theory," McGraw-Hill Book Company, New York.

Kohavi, Z., 1970, "Switching and finite automata theory," McGraw-Hill Book Company, New York, 1970.

Koutsoukos, X.D., Antsaklis, P.J., Stiver, J.A., and Lemmon, M.D., 2000, "Supervisory control of hybrid systems," Proceedings of the IEEE, Vol. 88, No. 7, pp. 1026-1049.

Lavagno, L., Keutzer, K., and Sangiovanni-Vincentelli, A., 1991, "Algorithms for synthesis of hazard-free asynchronous circuits," Proceedings of the 28th ACM/IEEE Design Automation Conference, San Francisco, CA, pp. 302-308.

Lavagno, L., Moon, C.W., and Sangiovanni-Vincentelli, A., 1994, "Efficient heuristic procedure for solving the state assignment problem for event-based specifications," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 14, pp. 45-60.

Lin, B., and Devadas, S., 1995, "Synthesis of hazard-free multilevel logic under multiple-input changes from binary decision diagrams," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 14, No. 8, pp. 974-985.

Lin, F., 1993, "Robust and adaptive supervisory control of discrete event systems," IEEE Transactions on Automatic Control, Vol. 38, No. 12, pp. 1848-1852.

Lin, F., and Wonham, W.M., 1990, "Decentralized control and coordination of discrete event systems with partial observation," IEEE Transactions on Automatic Control, Vol. 35, No. 12, pp. 1330-1337.

Liu, C., 1963, "A state variable assignment procedure for asynchronous sequential switching circuits," Journal of the Association of Computing Machinery, Vol. 10, pp. 209-216.

MacLane, S., and Birkhoff, G., 1967, "Algebra," The Macmillan Company, New York.

Mago, G., 1971, "Realization of methods for asynchronous sequential circuits," IEEE Transactions on Computers, C-20, pp. 290-298.

Maki, G., and Tracey, J., 1971, "A state assignment procedure for asynchronous sequential circuits," IEEE Transactions on Computers, C-20, pp. 666-668.

Marshall, A., Coates, B., and Siegel, F., 1994, "Designing an asynchronous communications chip," IEEE Design & Test of Computers, Vol. 11, No. 2, pp. 8-21.

Masuyama, H., and Yoshida, N., 1977, "design of fail-safe asynchronous sequential machines," Transactions of the Institute of Electronics and Communication, Vol. E60, No. 10, pp. 527-532.

McCluskey, E.J., 1963, "Fundamental mode and pulse mode sequential circuits," Proc. IFIP Congress 1962, International Conference on Information Processing, Munich, August 27-September 1, 1962, pp. 725-730, Cicely M. Popplewell (ed.), North Holland Publishing Company, Amsterdam.

McCluskey, E.J., 1965, "Introduction to the theory of switching circuits," McGraw-Hill Book Company, New York.

Mealy, G.H., 1955, "A method for synthesizing sequential circuits," Bell Systems Technical Journal, Vol. 34, pp. 1045-1079.

Moon, C.W., Stephan, P.R., and Brayton, R.K., 1991, "Synthesis of hazard-free asynchronous circuits from graphical specifications," IEEE International Conference on Computer-Aided Design, pp. 322-325.

Moore, E.F., 1956, "Gedanken-experiments on sequential machines," Automata Studies, Annals of Mathematical Studies, No. 34, Princeton University Press, Princeton, NJ.

Moore, S.W., Taylor, G.S., Cunningham, P.A., Mullins, R.D., and Robinson, P., 2000, "Self calibrating clocks for globally asynchronous locally synchronous systems," Proceedings of the International Conference on Computer Design, pp 73-78.

Murphy, T.E., 1996, "On the control of asynchronous sequential machines with races," Ph.D. Dissertation, Department of Electrical and Computer Engineering, University of Florida, Gainesville, Florida.

Murphy, T.E., Geng, X. J., and Hammer, J., 2002, "Controlling races in asynchronous sequential machines," Proceedings of the IFAC World Congress, Barcelona, July 2002.

Murphy, T.E., Geng, X. J., and Hammer, J., 2003, "On the control of asynchronous machines with races," IEEE Transactions on Automatic Control, Vol. 28, No. 6, pp. 1073-1081.

Nelson, R.J., 1968, "Introduction to automata," John Wiley and Sons, Inc., 1979.

Nishimura, N., 1990, "Asynchronous shared memory parallel computation," The 3rd ACM Symposium on Parallel Algorithms and Architectures SPAA '90, pp. 76-84.

Nishimura, N., 1995, "Efficient asynchronous simulation of a class of synchronous parallel algorithms," Journal of Computer and System Sciences, Vol. 50, No. 1, pp. 98-113.

Nowick, S.M., 1993, "Automatic synthesis of burst-mode asynchronous controllers," Ph.D. Dissertation, Department of Computer Science, Stanford University, Stanford, California.

Nowick, S.M., and Coates, B., 1994, "UCLOCK: Automated design of high-performance unclocked state machines," Proceedings of the IEEE International Conference on Computer Design (ICCD), pp. 434-441.

Nowick, S.M., Dean, M.E., Dill, D.L., and Horowitz, M., 1993, "The design of a high-performance cache controller: A case study in asynchronous synthesis," Proceedings of the 26th Hawaii International Conference on System Sciences, pp. 419-427.

Nowick, S.M., and Dill, D.L., 1991, "Synthesis of asynchronous state machines using a local clock," IEEE International Conference on Computer Design, pp. 192-197.

Oikonomou, K.N., 1992, "Abstractions of finite-state machines and immediately-detectable output faults," IEEE Transactions on Computers, Vol. 41, No. 3, pp. 325-338.

Oliviera, D.L., Strum, M., Wang, J.C., and Cunha, W.C., 2000, "Synthesis of high performance extended burst mode asynchronous state machines," Proceedings of the 13th Symposium on Integrated Circuits and Systems Design, pp. 41-46.

Ozveren, C.M., and Willsky, A.S., 1990, "Observability of discrete event dynamic systems," IEEE Transactions on Automatic Control, Vol. 35, No. 7, pp. 797-806.

Ozveren, C.M., Willsky, A.S., and Antsaklis, P.J., 1991, "Stability and stabilizability of discrete event dynamics systems," Journal of the Association of Computing Machinery, Vol. 38, No. 3, pp. 730-752.

Park, S-J., and Lim, J-T., 2002, "Robust and nonblocking supervisory control of nondeterministic discrete event systems using trajectory models," IEEE Transactions on Automatic Control, Vol. 47, No. 4, pp. 655-658.

Patterson, W.W., and Metze, G., 1974, "Fail safe asynchronous sequential machine," IEEE Transactions on Computers, Vol. C-23, No. 4, pp. 369-374.

Piguet, C., 1991, "Logic synthesis of race-free asynchronous CMOS circuits," IEEE Journal of Solid-State Circuits, Vol. 26, No. 3, pp. 371-380.

Plateau, B., and Atif, K., 1991, "Stochastic automata network of modeling parallel systems," IEEE Transactions on Software Engineering, Vol. 17, No. 10, pp. 1093-1108.

Ramadge, P.J., and Wonham, W.M., 1987, "Modular feedback logic for discrete event systems," SIAM Journal of Control and Optimization, Vol. 25, pp. 1202-1218.

Ramadge, P.J., and Wonham, W.M., 1989, "The control of discrete event systems," Proceedings of the IEEE, Vol. 77, No. 1, pp. 81-99.

Ramadge, P.J.G., and Wonham, W.M., 1987, "Supervisory control of a class of discrete event processes," SIAM Journal of Control and Optimization, Vol. 25, No. 1, pp. 206-230.

Sen, R.K., 1985, "On state assignment of asynchronous sequential machines," Proceedings – COMPINT 85: Computer Aided Technologies, pp. 433-440.

Schönfisch, B., and De Roos, A., 1999, "Synchronous and Asynchronous Updating in Cellular Automata", BioSystems, 51(3), pp. 123-143.

Shields, M.W., 1987, "An introduction to automata theory," Blackwell Scientific Publications, Boston, Massachusetts.

Thistle, J.G., and Wonham, W.M., 1994, "Control of infinite behavior of finite automata," SIAM Journal of Control and Optimization, Vol. 25, No. 1, pp. 206-230.

Tracey, J.H., 1966, "Internal state assignments for asynchronous sequential machines," IEEE Transactions on Electronic Computers, EC-15, pp. 551-560.

Turing, A., 1936-1937, "On computable numbers, with an application to the entscheidungs-problem," Proceedings of the London Mathematical Society, Ser. 2-42, pp. 230-265 with a correction, Ibid, Ser. 2-43, pp.544-546, 1936-1937.

Unger, S.H., 1959, "Hazards and delays in asynchronous sequential switching circuits," IRE Transactions on Circuit Theory, Vol. CT-6, No. 1, pp.12-25.

Unger, S.H., 1969, “Asynchronous sequential switching circuits,” Wiley Interscience, New York, NY.

Unger, S.H., 1977, “Self-synchronizing circuits and non-fundamental mode operation,” IEEE Transactions on Computers, Vol. 26, No. 3, pp. 278-281.

Unger, S.H., 1995, “Hazards, critical races and metastability,” IEEE Transactions on Computers, Vol. 44, No. 6, pp. 754-768.

Watanable, Y. and Brayton, R., 1993, “Computing permissible behaviors of FSM’s,” Proceedings of the International Conference on Computer-Aided Design, pp. 316-320.

Whitaker, S., and Maki, G., 1992, “Self synchronized asynchronous sequential pass transistor circuits,” IEEE Transactions on Computers, Vol. 41, No. 10, pp. 1344-1348.

Wolfram, S., 1994, “Cellular automata and complexity,” World Scientific Pub Co Inc., 1986.

Wonham, W.M., and Ramadge, P.J.G., 1987, “On the supremal controllable sublanguage of a given language,” SIAM Journal of Control and Optimization, Vol. 25, pp. 637-659.

Yoeli, M., and Rinon, S., 1964, “Application of ternary algebra to the study of static hazards,” Journal of the Association of Computing Machinery, Vol. 11, No. 1, pp. 84-97.

Yu, M.L., and Subrahmanyam, P.A., 1992, “A path-oriented approach for reducing hazards in asynchronous design,” Proceedings of the Design Automation Conference, 29th ACM/IEE, pp. 239-244.

Zambonelli, F., Roli, A., and Mamei, M., 2003, “Dissipative cellular automata as minimalist distributed systems : A study on emergent behaviors,” 11th IEEE EUROMICRO Conference on Parallel, Distributed, and Network Processing, Genova (I), IEEE CS Press, pp. 250-257.

BIOGRAPHICAL SKETCH

Niranjan Venkatraman was born in Bangalore, India, and obtained his Bachelor of Engineering (B.E.) degree in Electrical and Electronics Engineering in May 2000 from the College of Engineering, Guindy, Chennai, a part of Anna University. Since August 2000, he has been a Ph.D. student in the Department of Electrical and Computer Engineering at the University of Florida, Gainesville, FL. During the pursuit of his Ph.D., he obtained his M.S. in Electrical and Computer Engineering from the University of Florida in December 2002. His research interests include control theory, control systems, automata theory, and the application of control theory to asynchronous networks and parallel computation.