

AFRL-RI-RS-TR-2007-271
Final Technical Report
December 2007



LEARNING AND SELF-REPAIRING SYSTEMS

University of Florida

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Air Force Research Laboratory Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RI-RS-TR-2007-271 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/

/s/

CHRISTOPHER FLYNN
Work Unit Manager

JAMES A. COLLINS, Deputy Chief
Advanced Computing Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE				<i>Form Approved</i> OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.					
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) DEC 2007		2. REPORT TYPE Final		3. DATES COVERED (From - To) Jul 06 – Jun 07	
4. TITLE AND SUBTITLE LEARNING AND SELF-REPAIRING SYSTEM				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER FA8750-06-1-0175	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Jacob Hammer				5d. PROJECT NUMBER NBGQ	
				5e. TASK NUMBER 10	
				5f. WORK UNIT NUMBER 14	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Florida 319 Weil HL Gainesville FL 32611-5500				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/RITB 525 Brooks Rd Rome NY 13441-4505				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-RI-RS-TR-2007-271	
12. DISTRIBUTION AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# WPAFB 07-0686					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT The research covered by this grant concentrated on the development of computing algorithms for learning and self-repairing systems. During the report period, several existing and new methodologies were critically examined. The research resulted in the development of the concepts of learning blocks and association degree, which facilitate the development of learning and self-repairing systems. Methodologies based on these concepts allow a system to extract critical information from its past operation to automatically generate remedies for future malfunctions.					
15. SUBJECT TERMS Learning, Self-repairing systems, genetic algorithms					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UL	18. NUMBER OF PAGES 52	19a. NAME OF RESPONSIBLE PERSON Christopher Flynn
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (Include area code) N/A

TABLE OF CONTENTS

1. SUMMARY	1
2. THE DEMONSTRATION SYSTEM	2
3. THE OPTIMIZATION CRITERION	3
4. IMPLEMENTATION OF THE GENETIC ALGORITHM	4
5. LEARNING BLOCKS	6
6. THE LEARNING PROCESS AND THE ASSOCIATION DEGREE	9
7. ADAPTIVE CONTROL OF LEARNING SYSTEMS	11
8. PUBLICATIONS	14
9. CONCLUSIONS	15
10. APPENDIX: PUBLICATIONS	16

1. SUMMARY

The research that was conducted on this project fulfilled all the objectives set forward at the start of the project. We have developed a simulation system that demonstrates the advantages of our learning and self-repair methodology and we have initiated a theoretical study into the foundations of a general design methodology based on learning and self-repair. The theoretical foundation is built around a framework of clockless logic sequential machines, allowing the theory to be applied to extensively parallel computing systems.

a) Development and construction of a simulation system: We have a simulation system that demonstrates the advantages of learning and self repair over the existing technology of genetic algorithms. The system demonstrates in the clear visual way the disadvantages of genetic algorithms: the emergence of non-viable solutions during the process of chromosome breeding and mutation; the long time genetic algorithms take to approach optimal performance in high complexity systems; and the lack of learning, which requires the system to start from scratch after every significant failure. These disadvantages bring to light the superiority of the learning block approach we undertook, and point to the importance of continuing the development of learning methodologies. Here is a brief review of the setup of our genetic algorithm test framework.

The simulation test bed we used simulates the control of the traffic in a computer communication network. The simulation system was developed specifically for the simulation of self-repair algorithms for large scale distributed computing clusters, the self-repair of computer communication networks, and self-repair of difficulties in the cooperative operation of multiple combat units. The simulation system consists of nodes and links, where the nodes control the traffic flow through the links. In the current format, each node is connected to four branches that emerge from the node, where each branch allows bi-directional traffic. Four control handles are provided at each node, allowing to control the flow rate of each of the four exit gates at each node. The traffic flow rate through each gate can be controlled independently. In addition, each node generates traffic that enters the network at the node, simulating data, instructions, or computing results that originate at the node.

In the case of distributed computing clusters, each node represents a computing center of the cluster, while the links represent the flow of data, instructions, and results from one cluster to another. In the case of a computer communication network, the nodes represent routers and the links represent conduits of the network. In the case of cooperating combat units, the nodes represent individual units and the links represent the flow of assigned tasks among the units.

2. THE DEMONSTRATION SYSTEM

The demonstration program that was used includes 30 nodes and more than 50 links. Thus, it represents a distributed computing cluster with 30 computing centers. Each node is capable of emitting traffic into the network, representing the instructions and data generated at the computing cluster or computing network node it represents. In addition, each gate at each node has a queue, indicating elements (instructions or data packets) waiting to leave the node in the corresponding direction. All the algorithms that are implemented in the system are scalable, so larger computing clusters can be managed with the same algorithms.

The status of each node in the simulation is represented by a four digit string of digits (a *quadruplet*), where each digit represents the status of one of the four gates at the node: the first digit represents the west gate, the second digit represents the north gate, the third digit represents the east gate, and the fourth digit represents the south gate. Each digit is an integer between 0 and 5, where a digit c represents a flow rate of $0.2c$ per second, so that the flow rate varies between zero (closed gate) and 1 (fully open gate). For example, the string (0130) indicates that the west gate is closed, the north gate allows a flow of 0.2/sec, the east gate allows a flow of 0.6/sec, and the south gate is closed.

In addition, malfunctions at a node are represented by the character N which indicates that the corresponding gate of the node is permanently closed. Thus, the string (00N0) indicates that the east gate of the node is permanently closed; the west, north, and south gates are also closed at this time, but the algorithm can open them in its attempt to create a more efficient flow through the simulation. In this way, the status of the entire system is then represented by a string of 30 quadruplets; for example, the string (0240)(N501) (003N).

Each such string represents a possible control option for the traffic flow in our network. It is often referred to as a *chromosome*.

To implement a genetic algorithm, we start from an initial population S_0 of 4 chromosomes C_1 , C_2 , C_3 , and C_4 . Except for the locations of the characters N, which represent defective gates, the characters in each one of the four initial chromosomes are selected randomly.

3. THE OPTIMIZATION CRITERION:

When the simulation reaches a steady state, the criterion is taken as

$J :=$ number of queues that contain more than 5 elements in steady state.

Note that the values of J are integers.

Step G1: The system is run to steady state with each one of the four initial chromosomes, and the value of the criterion J is calculated in each case. This yields the four criterion values J_1, J_2, J_3 , and J_4 . (If one of the chromosomes does not yield as steady state, then it is discarded and replaced by a randomly selected new chromosome.)

Step G2: The next step of the genetic algorithm is the reproduction process. We produce multiple copies of our strings, generating more copies of the strings that yield a better criterion value. Since we are seeking a minimum in this case, the number of copies of string C_i is selected in proportion to $1/J_i$. For a real number a , let $[a]^t$ be the integer obtained by truncating a . We then create

$[100/J_1]^t$ copies of the chromosome C_1 ;

$[100/J_2]^t$ copies of the chromosome C_2 ;

$[100/J_3]^t$ copies of the chromosome C_3 ;

$[100/J_4]^t$ copies of the chromosome C_4 ;

This creates a set S_1 of chromosomes.

Step G3: The next step is the crossover step: We select randomly four pairs of chromosomes from the set S_1 . Let's denote the resulting pairs by (C_{11}, C_{21}) , (C_{12}, C_{22}) , (C_{13}, C_{23}) , (C_{14}, C_{24}) . For each one of these pairs, we select a random integer $1 \leq N_k \leq 30$, $k = 1, 2, 3, 4$. We then create from each pair a new chromosome, using the following process: for the pair (C_{1k}, C_{2k}) , we use the first N_k quadruplets of C_{1k} and the last $(30 - N_k)$ quadruplets of C_{2k} . Denote by C'_k the resulting chromosome, $k = 1, 2, 3, 4$. This yields a new set of four chromosomes.

Step G4: The next step is the mutation step: create a new string $S := C'_1 C'_2 C'_3 C'_4$ of 480 bits by concatenating the four chromosomes. Pick a random integer r in the range $[1, 10000]$. If $1 \leq r \leq 480$ and if position r in S is not N , then switch the value of character r of S according to the following table:

Original Value	Switch to
0	1

1	2
2	3
3	4
4	5
5	0
N	N

Finally, divide S back into 4 chromosomes $C_1^1, C_2^1, C_3^1, C_4^1$. On average, this process results in the change of one gate value every twenty runs. Repeat from Step G1.

This process continues until a maximal value of J is obtained.

4. IMPLEMENTATION OF THE GENETIC ALGORITHM

The genetic algorithm framework described above was implemented and run in a real time simulation, with the objective to optimize system performance after various failures. The simulations pointed to fundamental deficiencies of the genetic algorithm approach to the problem of controlling a large digital network that is subject to defects and failures.

1) One of the fundamental weaknesses of genetic algorithms in our simulations turned out to be the process of finding the initial four chromosomes. Recall that the initial four chromosomes C_1, C_2, C_3 , and C_4 are found through a random process. However, in our simulations, the vast majority of randomly constructed chromosomes did not lead to a steady state of the traffic in our simulated computer network. Instead, the vast majority of randomly selected chromosomes lead to conditions of network congestion and continually increasing queues at the network gates. Such chromosomes are, of course, not suitable for the operation of our traffic network, and they cannot be used as a basis of the process of breeding new chromosomes. They have been discarded, and new chromosomes have been selected and tested. In most cases, the simulation had to run through a rather large number of randomly selected chromosomes, before it was able to find four chromosomes that control the network to a steady state and do not induce congestion. Furthermore, the determination of steady state is a rather lengthy process - the system has to run for a period of time before it can be determined that steady state has been reached. As this process has to be repeated after every system failure, it is quite obvious that genetic algorithms do

not facilitate fast recovery of the computing network after failure, and hence are not suitable for the application at hand.

2) The process of breeding the chromosomes created similar difficulties. For the size of the network we are currently using (30 nodes and 50 links) and under high traffic conditions, it happened rather often that the process of randomly breeding two chromosomes did not yield a viable chromosome. In other words, very often chromosomes that resulted from the breeding process lead to traffic congestion in the network, and did not lead the network toward steady state traffic conditions. Thus, it seems that, for high complexity systems, the process of chromosome breeding can become dysfunctional. In such case, we had to return to the random selection process to create a new chromosome. The latter process, as indicated earlier, required a lengthy and time consuming process, as part of the attempt to find a chromosome that controls the traffic in the network without creating conditions of congestions and allows the network to reach a steady state.

3) The random mutation process that forms an integral part of the genetic algorithm process encountered similar difficulties. Sometimes, under heavy network traffic conditions, genetic mutation that was applied to a good chromosome resulted in a chromosome that caused conditions of traffic congestion within our computing network and had to be discarded. As before, such an occurrence forced us to default back to the random selection process in order to obtain the missing chromosome. As indicated earlier, the latter process was fraught with uncertainty, often yielding dysfunctional chromosomes.

4) Recall that one of the main objectives of this research project is to develop a control mechanism that allows the system to recover quickly after a significant malfunction. A significant malfunction in our case refers to a situation where 15% to 30% of the network's nodes become dysfunctional. To simulate such a situation, we randomly assign the character N indicating a permanently blocked node gate to a random fraction of the nodes, varying between 15% and 30%. In our case, the network has a total of 30 nodes; as each node has 4 gates, we have a total of 120 gates. To simulate failure, we block a random selection of 18 to 36 nodes out of the total of 120 gates, where the number of blocked nodes is selected randomly. The objective, of course, is to evaluate the capability of the genetic algorithm to return the system to proper function.

In this case as well, genetic algorithms do not seem to be an ideal choice for the task. After failure simulation was applied to our network, the existing chromosomes often did not perform properly, leading the network to conditions of traffic congestion. As a result, the existing chromosomes could not be used, and the entire chromosome generation process had to be re-run. As indicated earlier, this is a rather time consuming process.

It seems that genetic algorithms do not learn and do not accumulate the experience necessary to operate a system in the presence of failure. A genetic

algorithm does not seem to gather intimate knowledge about the system structure, knowledge that can be used to help the system recover quickly from significant malfunction. Genetic algorithms do not seem to learn the system's structure and do not seem capable of helping implement a quick recovery after major failure. On the contrary, attempts to control the system after significant failure by using the results of an earlier run of a genetic algorithms often seem to worsen the system's condition after failure rather than leading the system back to steady operation.

To summarize, it seems that genetic algorithms are not suitable for running systems that may vary significantly over the course of time, as is the case with systems prone to failure. Especially, for systems that are subject to significant failure and malfunction, genetic algorithms do not provide a general path to recovery. In most cases, after each malfunction or failure, the entire genetic algorithm process needs to re-run anew in order to return the computing network to proper function. Along this process, the process of breeding new genetic algorithms often destabilizes the system and leads to network congestion, resulting in an extended period of time during which the system is not operational. Thus, a new approach must be developed to allow complex systems to recover automatically from significant failure.

5. LEARNING BLOCKS

The learning block methodology that is being developed as part of this research project offers substantial advantages when it comes to overcoming the effects of failure on a complex system. With the learning block methodology, knowledge about the structure of the system is collected and accumulated, and this knowledge is automatically used by the system to overcome the effects of failures and defects. The learning system develops a database that relates various aspects of performance deterioration to the corrective actions taken in the past against such failures and malfunctions. This database allows the system to apply the correct remedy after a malfunction or failure with only a rare process of trial and error, and to return the network quickly to proper operation. This database is stored within subroutines of the main learning algorithm.

Before discussing the learning process to be used in our learning block simulation platform, we briefly review a few specifics about the simulation platform itself. Recall that our simulation platform consists of a computer communication network with 30 nodes and about 50 links. Each node has four gates, which control the flow of data and instructions into the four links that connect to the node. The state of the computing network is therefore characterized by a string of 120 characters, which each character describes the

status of one of the network's gates.

In this context, a learning block represents a list of gates with their corresponding transmission rates. In other words, each learning block assigns flow rates to a certain subset of gates in the network. To represent our learning blocks, we number the gates of our computing network and designate them by the characters $g_1, g_2, \dots, g_{120}$. Recalling that the flow rate through a gate is indicated by one of the integers 1, 2, 3, 4, 5 or by the character N which indicates a disabled gate, we combine these symbols so that, for example, the designation $g_7(3)$ indicates that gate 7 is open to flow rate 3, or $g_2(N)$ indicates that gate 2 is disabled. In these terms, a learning block is a string $g_a(j_a) g_b(j_b) \dots g_l(j_l)$ - it simply indicates the flow rates of the listed gates, and has no impact on other gates. Thus, for example, the string $g_3(4)g_6(2)g_{25}(5)$ indicates that the flow through gate 3 is set to 4; the flow through gate 2 is set to 6; and the flow through gate 25 is set to 5.

Comparing the learning blocks to chromosomes, recall that the chromosome in this case is a string of 120 characters, where each character describes the flow rate through one of the gates of the network. When a chromosome is activated, it impacts the flow through all the gates in the computing network. The learning block, on the other hand, affects only the gates that appear in its string, and has no impact on the other gates of the network.

Once a learning block is applied, we wait for the network to achieve steady state, and the impact of the learning block on the gate queues in the network is evaluated. Two outcomes are possible:

- 1) None of the queues increases in size and steady state is achieved.
- 2) Some queues continue or start to get bigger.

In case 1), the learning block is left in place and its impact is recorded; in case 2), the effect of the learning block on the various queues is recorded. This results in four lists of gates:

- a) gates whose queues continue to grow after the application of the tested learning block;
- b) gates whose queues started to grow immediately after the application of the tested learning block;
- c) gates whose queues remained unchanged by the tested learning block.
- d) gates whose queues declined after applying the tested learning block.

These four lists are then forwarded to the self-repair algorithm, which uses the lists to extract information about the topology of the network. Based on its analysis of current and previous outcomes, the algorithm then selects the next learning block to be applied to the system. This analysis is based on the notion of "association degree", which is described later in this report. In this way,

correction of faults becomes faster and more effective with the progress of time, as the self-repair algorithm learns the topology of the network from its analysis of the learning block outcomes.

The implications to our underlying applications are as follows. In the case of a computing cluster, assume that one of the computing centers of the cluster has become inoperable due to damage or malfunction, or that a link connecting two computing centers has been damaged. As a result, the queue of instructions and results forwarded to the inoperable computing center or to the damaged computing link starts growing; at some point, it will exceed its warning bound. At this point, a re-distribution of the flow of instructions between the remaining computing centers of the cluster is performed through the activation of a learning block. If the application of the learning block stops the growth of all queues in the network, then the problem has been corrected. The results are forwarded to the self-repair algorithm to extract structural information for future use. If the learning block does not mitigate the growth of all queues, then it is deactivated, and the results of its impact are forwarded to the self-repair algorithm for analysis. After analysis, the self-repair algorithm suggests another learning block for testing, based on its analysis of the network topology. This process improves the reaction speed and the response capabilities of the system through a learning process.

Similarly, in the case of a digital communication network, the self-repair algorithm starts when one or more of the packet queues at a router reach the warning bound. The algorithm then applies a learning block, which consists of a list of gates whose transmission rate is reassigned. The outcome of the learning block application is then analyzed by the self-repair algorithm, with the objective of extracting as much structural information as possible about the network. If desirable results have not been achieved, the learning block is removed, and another learning block is selected based on the analysis of previous outcomes. In this way, the self-repair algorithm learns and becomes able to predict the implications of applying a learning block, and the self-repair process becomes more and more efficient as time goes by. Similar implications hold for multiple cooperating combat units. The learning algorithm creates an association between each learning block and its impact on the status of the network queues. The algorithm singles out learning blocks whose impact is consistent under a given set of circumstances.

EXAMPLE 1. Referring to the 30 node network described earlier in this report, consider the following learning block.

Learning Block A: increase the flow through the west gate of node 12 by two steps; closes the south gate of node 18; reduces by one step the flow through the north gate of node 27.

Suppose that a malfunction occurs in the network, causing an accumulation in the queues of the following gates: the north gate of node 7; the east gate of node 21; and the south gate of node 25. Assume that Learning Block A is applied after

this malfunction, and that we notice an improvement in the queues at the east gate of node 21 and at the south gate of node 25. From this outcome, we can conclude that Learning Block A influences the traffic in links that affect the traffic flow to or from the east gate of node 21 and at the south gate of node 25. On the other hand, it does not seem to have an influence on the traffic in links that lead to the north gate of node 7. This, of course, is very valuable structural information, which is stored in the database of the learning system. In the future, when the system encounters a new malfunction that causes an increase in the queues of the east gate of node 21 and the south gate of node 25, the system will attempt to correct the problem by applying Learning Block A.

As the system continues to operate and to handle more and more failures and malfunctions in the network, it accumulates more and more relationships in its database, relationships that associate each learning block with the malfunctions it can correct. The more malfunctions the system has experienced, the more precise the associations between learning blocks and their corrective effects become, and the more capable the system becomes in administering a quick recovery after malfunction.

6. THE LEARNING PROCESS AND THE ASSOCIATION DEGREE

The learning process is based on a weighting procedure that assigns greater weight to learning blocks that correct a given problem more often. Consider the following example. To simplify the wording, we will denote each buffer by a number and a letter, where the number indicates the node number and the letter indicates the direction of the buffer's gate. Recall that in our simulation scheme, every node has four gates - a north gate (N), and east gate (E), a south gate (S), and a west gate (W). It is convenient to denote every gate by a digit and a letter in the form nX , where n is the node number of the gate and X is one of the letters N, E, S, and W. In this notation, 12E indicates gate east of node 12; 20N designates the north gate of node 20. Assume then that the system went through two malfunctions:

Malfunction 1: the queues of gates 7S and 15E increase;

Malfunction 2: the queues of gates 7S and 23N increase.

To overcome the impact of these malfunctions, the system attempts two learning blocks - Learning Block B and Learning Block D, with the outcomes:

Learning Block B stops the queue growth in gate 7S in Malfunction 1 and in Malfunction 2; Learning Block D stops the queue growth in gates 7S and 23N in

Malfunction 2 only, and has no effect in the case of Malfunction 1.

These outcomes create a stronger association between Learning Block B and queue growth in gate 7S, since Learning Block B impacted this queue in both malfunctions.

To formalize this process, we define the novel concept of the *association degree*. The association degree assigns an integer to each combination of a building block and a network gate, as follows. In formal terms, a pair (Y, nX) refers to the learning block Y and the gate nX . For example, the pair $(C, 9N)$ refers to Learning Block C and the north gate of node 9 .

The *association degree* assigns an integer to each pair (Y, nX) . The initial value of this integer is zero, and it is adjusted after analyzing the outcome of applying learning block Y to a failure that involves the queue at the gate nX . The association degree of (Y, nX) is increased by one after every distinct failure pattern for which Learning Block Y reduces the queue at gate X of node n . It is reduced by one if learning block Y causes an increase in the queue length at gate X of node n ; and it is left unchanged when learning block Y has no effect on the queue length at gate X of node n .

EXAMPLE 2. Consider the situation described earlier in Example 1. Here, after the two failure incidents, the association degree of the pair $(B, 7S)$ is 2, whereas the association degree of the pair $(D, 7S)$ is 1.

The association degree is a dynamical quantity that changes and evolves as the system experiences its malfunctions. Its value points to the most likely learning block for overcoming a given malfunction. Intuitively speaking, the association degree is a quantitative summary of the experience gained by the system; a record of what the system has learned from its experience.

As the system continues its operation and encounters new malfunctions and failures, it examines the effects of the malfunction on the queues of the various gates and generates a list of the gates affected by the malfunction. Then, it examines its current list of association degrees, and chooses a learning block that shows a high association degree with the gates affected by the malfunction. The system then applies the selected learning block and examines the outcome. If the outcome is satisfactory and all queues have been stabilized, then the system updates the association degrees and continues its regular operation. If the outcome is not satisfactory, then the system updates the association degrees and applies another learning block, choosing from among the remaining blocks having highest association degree with the affected queues. This process continues until the system compensates for the effects of the latest malfunction. The association degree updates generated during this process will improve the reaction of the system in future malfunctions.

When compared to genetic algorithms, it is quite clear that the learning block approach provides faster and more effective corrections to system malfunctions.

When attempting a correction to a malfunction using the learning block approach, the system does not just apply a random remedy; it applies a remedy that has worked well in similar situations in the past. Therefore, the probability that the applied learning block provides an effective remedy is high, and this probability increases as the system gains more and more experience correcting malfunctions. The association degree is, therefore, a simple and effective tool for designing systems capable of learning and self-repair.

7. ADAPTIVE CONTROL OF LEARNING SYSTEMS

This part of our research deals with the development of a general theoretical framework for the design and implementation of learning and self-repairing systems. The system under consideration is represented by an asynchronous sequential machine. An asynchronous sequential machine, sometimes referred to as a clockless logic circuit, represents the fastest computing machines. This model captures a broad range of applications, including distributed computing clusters, highly distributed and extensively parallel computing systems, asynchronous computer communication networks, and the operation of multiple combat units.

More specifically, when considering distributed computing clusters, each computing center in the cluster operates independently, directing instructions and computational data to other members of the cluster in an asynchronous fashion. Data, results, and instructions are generated and transmitted by each computing center when required by its internal processing, without regard to the status of other members of the computing cluster. This indicates that the various members of a computing cluster are working in an asynchronous fashion. Asynchronous operation has the advantage of providing the highest possible speed, as computing centers do not have to wait for timing cues arriving from other computing centers. Furthermore, in cases of high performance computing, each computing center consists of a large aggregate of parallel processors working asynchronously. Thus, an asynchronous machine model is the most appropriate representation of this application area.

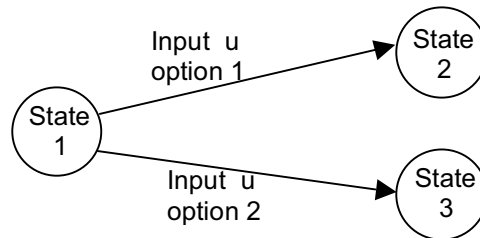
Regarding computer communication networks, it is widely known that asynchronous networks provide a higher throughput than synchronous networks, as there is no need to generate and wait for timing pulses; the network operates at the highest speed possible for its components. Consequently, an asynchronous machine model is most suitable for the representation of high-throughput computer communication networks.

Finally, regarding the operation of multiple combat units, these most often result in asynchronous operation as well, since each unit engages the enemy

and completes its tasks on its own terms, independently of the momentary operation of other units. Consider, for example, a concerted attack on a distributed range of mountain caves in enemy territory. Multiple attacks are initiated by independent units on each cave cluster; obviously, the destruction of one cave cluster occurs independently of the destruction of another cave cluster, as some of the cave clusters may be better protected or better built than others. Each unit then completes its task, without synchronizing the completion with other units. Thus, the operation of multiple combat units can be considered as an asynchronous system; our objective is to control this asynchronous system to maximize its effectiveness.

To represent faults and defects in the underlying application systems, we allow our asynchronous machine model to have several unknown transitions. For the unknown transitions, the outcome is not precisely known; instead, several options for the transition are provided. Here is an example of a transition with two options:

EXAMPLE 3. Assume that when the machine is in State 1 and the input character u is applied, two outcome options are possible: the outcome is either a transition to State 2 or a transition to State 3. Note that the machine is deterministic, so that once the outcome of the transition has been determined experimentally, the same outcome will be repeated in any future activation of the input character u at State 1. The situation can be represented by the following state diagram.



To represent an asynchronous machine with unknown transitions, we use the following *transition matrix*. Assume that the machine has n states, say $x^1, x^2, \dots, x^n$. We build an $n \times n$ matrix in which each row and each column correspond to a state. Thus, column 1 and row 1 correspond to the state x^1 ; column 2 and row 2 correspond to the state x^2 ; and so on. Then, entry i, j of the matrix consists of all input characters that induce a one step transition from the state x^i to the state x^j (i.e., from the state corresponding to the row to the state corresponding to the column). An input character that induces an unknown transition from a given state will appear several times in the row corresponding to the state of origin. Thus, the situation described in Example 3 is represented by the matrix

$$\begin{pmatrix} \square & u & u & \dots & \square \\ \square & \square & \square & \dots & \square \\ \square & \square & \square & \dots & \square \\ \square & \square & \square & \dots & \square \\ \square & \square & \square & \dots & \square \end{pmatrix},$$

where the squares $\square$ represent input values not considered in this example. As we can see, the input character u appears twice in row 1: once to represent a possible transition from the state x^1 to the state x^2 , and once to represent a possible transition from the state x^1 to the state x^3 , as indicated in the earlier state diagram.

If there is no transition possible from the state x^i to the state x^j , then we enter the character N in position i,j , where N indicates the lack of a transition (N is not a character of the input alphabet).

An important step in the development of learning and self-repair algorithms for an asynchronous machine is the partition of its state set into subsets $S := \{S_1, S_2, \dots, S_k\}$ of states that can be reached from each other through a known transition. Specifically, for any subset S_i , all pairs of states within S_i are connected through at least one specified transition. In other words, transitions within each one of the sets S_i are known and specified at the present time. On the other hand, transitions from one subset to another, say transitions from states of the subset S_i to states of the subset S_j , $i \neq j$, if possible, involve an unknown (or unpredictable) transition. As the learning and self-repair process revolves only around the management of unpredictable transitions, this partition of the state space results in a substantial reduction of computational complexity of the learning algorithms.

The partition S discussed in the previous paragraph is dynamic, namely, it changes as the system learns the outcome of more and more unpredictable transitions. In fact, the learning process of the machine is represented by the evolution of the partition S . The number of elements in S becomes smaller and smaller as the machine learns the outcome of more and more unpredictable transitions.

Recall that an asynchronous machine is described by an equation of the form

$$x_{k+1} = f(x_k, u_k),$$

where x_k is the state of the machine at step k , and u_k is the input of the machine at step k . To represent this machine in our theoretical framework, we have developed the *one-step transition matrix*, which was described earlier. Let $x^1, x^2, \dots, x^n$ be the states of the machine. The one-step transition matrix is then an $n \times n$ matrix in which each row and each column correspond to a state. Thus, column 1 and row 1 correspond to the state x^1 ; column 2 and row 2 correspond to the state x^2 ; and so on. Then, entry i, j of the matrix consists of all input characters that induce a one step transition from the state x^i to the state

x^j (i.e., from the state corresponding to the row to the state corresponding to the column).

EXAMPLE 4. Consider a machine with two states, x^1 and x^2 . Suppose the machine accepts two input characters a and b . Then, the one step reachability matrix

$$\begin{pmatrix} a & b \\ b & a \end{pmatrix}$$

indicates the following: the input character a leaves the machine in the state x^1 and in the state x^2 ; the input character b takes the machine from the state x^1 to the state x^2 and from the state x^2 to the state x^1 .

Assume further that the machine is affected by a malfunction, as a result of which some of the transitions are not known. In the above example, suppose we don't know exactly what transitions the input character a causes when the machine is in the state x^1 . For example, suppose, when the machine is in state 1, the input character a either leaves the machine in the state x^1 or causes a transition to the state x^2 . In such case, the one-step transition matrix takes the form:

$$\begin{pmatrix} a & a, b \\ b & a \end{pmatrix}.$$

The matrix indicates that, when the machine is in the state x^1 , the following options are possible: when the input character a is applied, the resulting transition might be either to the state x^1 or to the state x^2 ; if the character b is applied, the system moves to the state x^2 . The remaining transitions are as before. Thus, the one-step transition matrix is a very convenient way to represent uncertainties in the function of an asynchronous machine, uncertainties created by malfunctions or lack of precise information about the machine.

Dealing with one-step transition matrices that include uncertainties requires a special mathematical framework, which has been developed as part of the current research project. This framework is described in the following publications, which are attached as part of this report.

8. PUBLICATIONS

The following publications report about material developed in this project.

J. M. Yang and J. Hammer

[2007] "Counteracting the Effects of Adversarial Inputs on Asynchronous Sequential Machines", submitted for publication.

J. M. Yang and J. Hammer

[2007] "State Feedback Control of Asynchronous Sequential Machines with Adversarial Inputs", submitted for publication.

Copies of these publications are provided in the Appendix.

9. CONCLUSIONS

The present report presents a summary of the research conducted under Grant number FA8750-06-1-0175. In the course of this research, we introduced a new methodology for the design of engineering systems: the learning block methodology. This methodology forms a theoretical foundation for the development of equipment and systems capable of learning and self repair. During the course of the research, other methodologies that could potentially be used for the design of learning and self repairing systems were examined and compared to the learning block methodology. Specifically, significant effort was devoted to the examination of genetic algorithms in this context. The conclusion of this effort was that genetic algorithms do not form a suitable foundation for learning and self repairing systems.

The research performed under this grant shows that the learning block methodology forms a solid and efficient foundation for the design of learning and self repairing systems. Further research in this direction would explore the application of this methodology as a supervisory tool for specific critical systems, including flight control systems and large-scale distributed computing clusters. The studies conducted under the project reported here indicate that the learning block methodology can help improve the reliability and failure-endurance of such systems but at least an order of magnitude.

10. APPENDIX: PUBLICATIONS

This appendix includes copies of the following two publications:

J. M. Yang and J. Hammer

[2007] "Counteracting the Effects of Adversarial Inputs on Asynchronous Sequential Machines", submitted for publication.

J. M. Yang and J. Hammer

[2007] "State Feedback Control of Asynchronous Sequential Machines with Adversarial Inputs", submitted for publication.

Hard copies of the publications are attached. The publications are also included in the pdf version of this report.

STATE FEEDBACK CONTROL OF ASYNCHRONOUS SEQUENTIAL MACHINES WITH ADVERSARIAL INPUTS

by

Jung-Min Yang[†] and Jacob Hammer[‡]

ABSTRACT

The problem of controlling an asynchronous sequential machine in the presence of adversarial inputs is considered. Here, an adversarial input is an unknown and unauthorized input that attempts to interfere with the operation of the machine. The objective is to develop automatic state feedback controllers that counteract the effects of the adversarial input and restore desirable behavior to the controlled machine. Necessary and sufficient conditions for the existence of such controllers are presented in terms of an inequality condition between two numerical matrices. Whenever a controller exists, an algorithm for its design is provided.

June 2007

[†]Department of Electrical Engineering, Catholic University of Daegu, 330 Kumrak, Hayang, Kyongsan, Kyongbuk, 712-702, Republic of Korea. The work of this author was supported by the Research Grants of Catholic University of Daegu in 2006.

[‡]Department of Electrical and Computer Engineering, University of Florida, P.O. Box 116130, Gainesville, FL 32611, USA. The work of this author was supported in part by the US Air Force, through grant number FA8750-06-1-0175.

Counteracting the Effects of Adversarial Inputs on Asynchronous Sequential Machines

Jung-Min Yang* and Jacob Hammer**

*Department of Electrical Engineering, Catholic University of Daegu, 330 Kumrak, Hayang, Kyongsan, Kyongbuk, 712-702, Republic of Korea; e-mail: jmyang@cu.ac.kr

**Department of Electrical and Computer Engineering, University of Florida, P.O. Box 116130, Gainesville, FL 32611, USA; e-mail: hammer@mst.ufl.edu

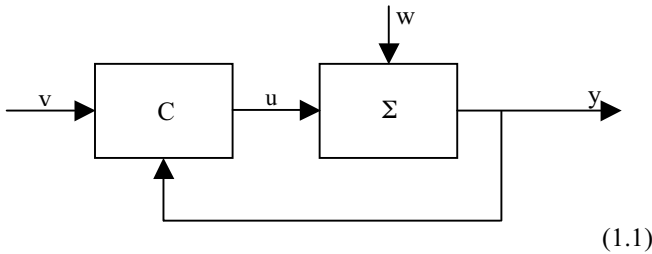
**The work of this author was supported in part by the US Air Force, grant number FA8750-06-1-0175.

Abstract: The problem of counteracting the effects of adversarial inputs on the operation of an asynchronous sequential machine is considered. The objective is to build an automatic state-feedback controller that returns an asynchronous sequential machine to its original state, after the machine has undergone a state transition caused by an adversarial input. It is shown that the existence of such a controller depends on certain reachability and detectability properties of the affected machine.

1. INTRODUCTION

In modern computing systems, one often encounters unauthorized and adversarial input agents that attempt to interfere with the proper operation of the system. We address the question of how a computing system can be made immune to such interferences. Our approach is based on automatic control: we deploy feedback controllers that take corrective action whenever an adversarial input attempts to affect the operation of the underlying computing system.

Specifically, we consider asynchronous sequential machines with two inputs: an input for controlling the machine (the *control input*), and an input used by an adversarial agent (the *adversarial input*). The control diagram is as follows.



Here, Σ is the asynchronous sequential machine being controlled, and C is another asynchronous machine serving as the controller. The control input of Σ is u and the adversarial input is w . The controller C counteracts action at w , making it possible for the closed loop machine to operate without interference. Necessary and sufficient conditions for the existence of C are presented in section 6, which also includes a description of the controller's structure. The closed loop machine of the diagram is denoted by $\Sigma_c(v, w)$, with v being the external command input.

Recall that an asynchronous sequential machine can be in a stable state or in a transient state. At a stable state, the machine dwells indefinitely until the input character is

changed. Transient states are traversed by the machine very quickly (ideally, in zero time), and are imperceptible by the user. Thus, when counteracting the effects of an adversarial input, it is only necessary to eliminate the effects on stable states. The operation of the controller C is, in fact, based on this principle: when the adversarial input causes a state transition of Σ , the controller turns the new state into a transient state of the closed loop machine, and returns Σ to the stable state it had before the interference. Thus, Σ resumes its original state very quickly (ideally, in zero time), and the effect of the adversarial input is eliminated.

Our discussion is within the framework developed by MURPHY, GENG, and HAMMER [2002 and 2003], GENG and HAMMER [2004 and 2005], and VENKATRAMAN and HAMMER [2006a, b, and c]. Studies dealing with other aspects of discrete event systems can be found in RAMADGE and WONHAM [1987], HAMMER [1994, 1995, 1996a, 1996b, 1997], DIBENEDETTO, SALDANHA, and SANGIOVANNI-VINCENTELLI [1994], THISTLE and WONHAM [1994], BARRETT and LAFORTUNE [1998], the references cited in these papers, and others. These publications do not address issues peculiar to the operation of asynchronous machines, such as the avoidance of critical races and the distinction between stable and transient states.

2. NOTATION AND BASICS

The machines we consider have a control input and an adversarial input, and they provide their state as output. Such machines are represented by a triplet $(A \times B, X, f)$, where A is the control input alphabet, B is the adversarial input alphabet, X is a set of states, and $f: X \times A \times B \rightarrow X$ is the *recursion function*. The operation is according to

$$x_{k+1} = f(x_k, u_k, w_k); \quad (2.1)$$

here, $u_0, u_1, u_2, \dots$ is the *control input sequence*; $w_0, w_1, w_2, \dots$ is the *adversarial input sequence*; and $x_0, x_1, x_2, \dots$ is the

sequence of the machine's states. The *step counter* k advances by one at a change of the machine's inputs or state.

(2.2) EXAMPLE. An asynchronous machine Σ with the control input alphabet $A = \{a, b\}$; the adversarial input alphabet $B = \{\alpha, \beta\}$; and the state set $X = \{x^1, x^2, x^3\}$. The recursion function f is described by the transition table:

state	(a, α)	(a, β)	(b, α)	(b, β)
x^1	x^1	x^1	x^1	x^2
x^2	x^3	x^1	x^2	x^2
x^3	x^3	x^3	x^1	x^2

A triplet (x, u, w) is a *stable combination* if $x = f(x, u, w)$, i.e., if the state x is a fixed point of the function f . A machine lingers at a stable combination until an input changes. A triplet (x, u, w) that is not a stable combination starts a chain of transitions $x_1 = f(x, u, w)$, $x_2 = f(x_1, u, w)$, ... If this chain terminates, then there is an integer $q \geq 1$ for which $x_q = f(x_q, u, w)$; then, (x_q, u, w) is a stable combination and x_q is the *next stable state*. If this chain of transitions does not terminate, then (x, u, w) is part of an *infinite cycle*. In this paper, we consider only machines with no infinite cycles; thus, every triplet (x, u, w) has a next stable state, as follows.

(2.3) LEMMA. In an asynchronous machine without infinite cycles, there is always a next stable state. ♦

To prevent unpredictable outcomes, it is common to enforce a policy where only one variable of an asynchronous machine is allowed change value at any instant of time (e.g., KOHAVI [1970]). This is referred to as *fundamental mode operation*. All the machines in this paper operate in fundamental mode.

(2.4) DEFINITION. An asynchronous machine Σ *operates in fundamental mode* if its inputs change value only when Σ is in a stable combination, and then at most one at a time. ♦

In fundamental mode operation of the configuration (1.1), only one of the machines Σ or C can undergo transitions at any instant of time. This leads us to:

(2.5) PROPOSITION. Configuration (1.1) operates in fundamental mode if and only if the following hold:

- (i) C is in a stable combination while Σ undergoes transitions, and Σ is in a stable combination while C undergoes transitions.
- (ii) The inputs u, w , and v change only while Σ and C are in a stable combination, and then only one at a time. ♦

Thus, the controller C must be designed so that (i) it commences transitions only after verifying that Σ is in a stable combination, and (ii) it adopts a stable combination before inducing a change in the input of Σ . This assures that the closed loop system is unambiguous and deterministic. As transitions of asynchronous machines are very quick (ideally, in zero time), fundamental mode operation is not restrictive.

3. ADVERSARIAL INPUTS

In general, the adversarial input character w_k is not

specified; it is only known that it belongs to a specified subset $v \subset B$ called the *adversarial uncertainty*. To include this information, we write $\Sigma = (A \times B, X, f, v)$. Starting from the initial state x_0 and applying the control input character u_0 , the next state of Σ can be any member of the set

$$f[x_0 \times u_0 \times v] := \bigcup_{w \in v} f(x_0, u_0, w) \subset X.$$

To describe stable transitions of the machine Σ , let x' be the next stable state of (x, u, w) . The *stable recursion function* s is defined by setting $s(x, u, w) := x'$. Considering adversarial uncertainty, all possible next stable states form the set

$$s^v(x, u) := s[x, u, v] = \{s(x, u, w) : w \in v\} \subset X. \quad (3.1)$$

4. DETECTABILITY AND REACHABILITY

By Proposition 2.5, fundamental mode operation requires the controller C to remain in a stable combination until Σ has reached its next stable state. To examine the conditions under which such a controller can be implemented, let w be an adversarial input character. Assume that Σ is in a stable combination at the state x , when the control input changes to u . Then, Σ embarks on the string of transitions

$$\theta(x, u, w) := \{x_1 := f(x, u, w), x_2 := f(x_1, u, w), \dots, x_{i(u, w)} := f(x_{i(u, w)-1}, u, w)\}, \quad (4.1)$$

where $x_{i(u, w)}$ is the next stable state. The set of all transition strings consistent with the adversarial uncertainty v is:

$$\theta[x, u, v] := \{\theta(x, u, w) : w \in v\}. \quad (4.2)$$

The next notion characterizes our ability to determine by state feedback whether or not Σ has reached its next stable state.

(4.3) DEFINITION. Let Σ be in a stable combination with the state x , when the control input character changes to u . The pair (x, u) is *detectable* if it is possible to determine by state feedback whether Σ has reached its next stable state. ♦

Here is a test to determine whether a pair is detectable.

(4.4) THEOREM. Let Σ be in a stable combination with the state x , when the control input character changes to u . Then, (i) and (ii) are equivalent for adversarial uncertainty ε :

- (i) The pair (x, u) is detectable.
- (ii) States of the set $s^\varepsilon(x, u)$ appear only at the end of strings belonging to $\theta[x, u, \varepsilon]$.

Proof (sketch). Consider a string $\theta(x, u, w) = \{x_0, x_1, x_2, \dots, x_{i(u, w)}\}$, where $w \in \varepsilon$, and assume, by contradiction, that (ii) is not valid. Then, $x_j \in s^\varepsilon(x, u)$ for an integer $0 \leq j < i(u, w)$, so that (x_j, u, w) is a transient combination, since $j < i(u, w)$. The inclusion $x_j \in s^\varepsilon(x, u)$ implies that there is an adversarial input character $w' \in \varepsilon$ for which (x_j, u, w') is a stable combination. Thus, x_j is a transient state in (x_j, u, w) while being a stable state in (x_j, u, w') , so that, at the state x_j , one cannot tell whether Σ is in a stable state. Whence, (i) implies (ii). Conversely, if every state $x' \in s^\varepsilon(x, u)$ appears only at the end of a string in $\theta[x, u, \varepsilon]$, then x' is always a stable state of Σ , and (ii) implies (i). ♦

Thus, we can determine whether Σ has reached its next stable state by checking if the current state of Σ is in $s^t(x, u)$. Next, we adapt to our present setting the following notion from MURPHY, GENG, and HAMMER [2002] and [2003].

(4.5) DEFINITION. Let Σ be an asynchronous machine with the state set $X = \{x^1, x^2, \dots, x^n\}$ and the stable transition function s . Let $w \in B$ be an adversarial input character. The *one-step matrix of stable transitions* $\rho(\Sigma, w)$ of Σ is an $n \times n$ matrix whose (i, j) entry $\rho_{ij}(\Sigma, w)$ consists of all pairs $w|u$, where u is a control input character satisfying $x^j = s(x^i, u, w)$; if there is no such u , then $\rho_{ij}(\Sigma, w) := w|N$, where N is a character not in A or in B :

$$\rho_{ij}(\Sigma, w) = \begin{cases} w|N & \text{if } \{u \in A : x^j = s(x^i, u, w)\} = \emptyset, \\ \{w|u : u \in A \text{ and } x^j = s(x^i, u, w)\} & \text{otherwise.} \end{cases} \blacklozenge$$

(4.6) EXAMPLE. For the machine Σ of Example 2.2,

$$\rho(\Sigma, \alpha) = \begin{pmatrix} \{\alpha|a, \alpha|b\} & \{\alpha|N\} & \{\alpha|N\} \\ \{\alpha|N\} & \{\alpha|b\} & \{\alpha|a\} \\ \{\alpha|b\} & \{\alpha|N\} & \{\alpha|a\} \end{pmatrix} \blacklozenge$$

To work with $\rho(\Sigma, w)$, we use the following projections: (A^+ is the set of all non-empty strings of characters of A .)

$$\begin{aligned} \Pi_a w|u &:= \begin{cases} w & \text{if } u \neq N, \\ \emptyset & \text{else,} \end{cases} \quad (\text{onto adversarial value}), \text{ and} \\ \Pi_c w|u &:= u \text{ for all } w|u \in B|(A^+ \cup N) \text{ (onto control value).} \end{aligned}$$

For two sets of strings $s_1, s_2 \subset B|(A^+ \cup N)$, we define an operation $s_1 \vee s_2$ akin to union by using the difference set

$$s_1 \vee s_2 := [s_1 \cup s_2] \setminus s_N,$$

where s_N is the set of all elements $w|N \in s_1 \cup s_2$ for which $[w|A^+] \cap [s_1 \cup s_2] \neq \emptyset$, i.e., all elements that appear both with N and non- N control input strings. Next, *concatenation* of strings $w_1|u_1, w_2|u_2 \in B|(A^+ \cup N)$ is given by

$$\text{conc}(w_1|u_1, w_2|u_2) := \begin{cases} w_1|u_1u_2 & \text{if } w_1 = w_2 \text{ and } u_1, u_2 \neq N, \\ w_1|N, w_2|N & \text{otherwise.} \end{cases}$$

For subsets of strings $\sigma_1, \sigma_2 \subset B|(A^+ \cup N)$:

$$\text{conc}(\sigma_1, \sigma_2) := \vee_{s_1 \in \sigma_1, s_2 \in \sigma_2} \text{conc}(s_1, s_2).$$

Now, define an operation similar to matrix multiplication for two $n \times n$ matrices P, Q with entries in $B|(A^+ \cup N)$:

$$(PQ)_{ij} := \vee_{k=1, \dots, n} \text{conc}(P_{ik}, Q_{kj}), \quad i, j = 1, 2, \dots, n.$$

We can use the powers $\rho^k(\Sigma, w) = \rho^{k-1}(\Sigma, w)\rho(\Sigma, w)$, $k = 1, 2, \dots$. The i, j entry of $\rho^k(\Sigma, w)$ consists of all strings $w|u$ that take Σ from a stable combination with the state x^i to a stable combination with the state x^j in exactly k steps; if there is no such string, then $u = N$.

(4.7) EXAMPLE. Continuing Example 4.6:

$$\rho^2(\Sigma, \alpha) = \begin{pmatrix} \{\alpha|aa, \alpha|ab, \alpha|ba, \alpha|bb\} & \{\alpha|N\} & \{\alpha|N\} \\ \{\alpha|ab\} & \{\alpha|bb\} & \{\alpha|ba, \alpha|aa\} \\ \{\alpha|ab, \alpha|ba, \alpha|bb\} & \{\alpha|N\} & \{\alpha|aa\} \end{pmatrix} \blacklozenge$$

The *matrix of m stable transitions* of Σ is defined by

$$R(m, \Sigma, w) := \vee_{i=1, \dots, m} \rho^i(\Sigma, w); \quad (4.8)$$

it characterizes all the transitions that can be accomplished in m or fewer stable steps. Allowing m to grow indefinitely yields the *extended matrix of stable transitions* $R^*(\Sigma, w) := \vee_{i \geq 1} \rho^i(\Sigma, w)$ that characterizes all stable transitions of Σ . The next statement resembles MURPHY, GENG, and HAMMER [2003, Proposition 3.9].

(4.9) LEMMA. The following are equivalent for all integers $m \geq n-1$ and all $i, j = 1, 2, \dots, n$.

- (i) The entry $R_{ij}(m, \Sigma, w)$ includes a string $w|u$ with $u \neq N$.
- (ii) The entry $R_{ij}^*(\Sigma, w)$ includes a string $w|u$ with $u \neq N$. $\blacklozenge$

Thus, when $m \geq n-1$, the matrix $R(m, \Sigma, w)$ characterizes all stable transitions of Σ for the adversarial input character w .

(4.10) DEFINITION. For the adversarial input uncertainty v , the *one-step stable transitions matrix* $\rho(\Sigma, v)$ is an $n \times n$ matrix with entries $\rho_{ij}(\Sigma, v) := \vee_{w \in v} \rho_{ij}(\Sigma, w)$, $i, j = 1, 2, \dots, n$. $\blacklozenge$

The matrix $\rho(\Sigma, v)$ includes all one-step stable transitions that are compatible with an adversarial input character in v .

5. COMPLETE SETS OF STRINGS

The next notion is critical for feedback control (compare to VENKATRAMAN and HAMMER [2006a, b, c]).

(5.1) DEFINITION. Let Σ be an asynchronous machine with the adversarial uncertainty v , and let x^i, x^j be states of Σ . There is a *feedback path* from x^i to x^j if there is a state feedback controller that takes Σ from a stable combination with x^i to a stable combination with x^j in fundamental mode, given only that the adversarial input is within v . $\blacklozenge$

Below, we develop a test to determine whether there is a feedback path from x^i to x^j . If a feedback path exists, then an automatic controller can undo undesirable transitions from x^j to x^i . Note that, due to fundamental mode operation, the adversarial input remains constant along a feedback path.

Adversarial uncertainty may decline along a feedback path, since the machine's response provides information about the adversarial input. For example, let the adversarial uncertainty be $v = \{w^1, w^2\}$, and let s be the stable recursion function of Σ . Assume that Σ is at a stable combination with the state x and the control input character u , when the control input changes to u' . We have two options for the next stable state:

$$\begin{aligned} x' &:= s(x, u', w^1) \text{ when the adversarial input character is } w^1; \\ x'' &:= s(x, u', w^2) \text{ when the adversarial input character is } w^2. \end{aligned}$$

Clearly, if $x' \neq x''$, then we can determine the adversarial input character from the next stable state, resolving the uncertainty. Thus, the adversarial uncertainty may decline along a feedback path. To discuss the general case, we need some notation. Let $S \subset B|A^+$ be the set of all strings that take Σ from a stable combination with the state x to a stable combination with the state x' , i.e., all strings $w|u =$

$w|u_0u_1\dots \in B|A^+$ for which $s(x,u,w) = x'$. For a string $\sigma = w|u_0u_1\dots u_k \in S$ and an integer $q \geq 0$, denote

$$\sigma|_q := \begin{cases} w|u_0u_1\dots u_q & \text{if } q \leq k, \\ w|u_0u_1\dots u_k & \text{if } q > k. \end{cases}$$

The string $\sigma|_q$ takes Σ to a stable combination with the state $x_q := s(x, \sigma|_q) := s(x, u_0u_1\dots u_q, w)$, passing through the stable states $x_0(\sigma) := s(x, \sigma|_0)$, $x_1(\sigma) := s(x, \sigma|_1)$, ..., $x_q(\sigma) := s(x, \sigma|_q)$, where $x_0(\sigma) = x$ and $x_k(\sigma) = x'$.

For a string $\sigma = w|u_0u_1\dots u_k \in S$, denote

$$\Pi^p \sigma := \begin{cases} u_p & \text{for } p = 0, 1, \dots, k, \\ u_k & \text{for all } p > k. \end{cases}$$

Now, let Σ be in a stable combination with the state z when the control input value changes to u , and let z'' be the next stable state of Σ . The set of all adversarial input characters $w \in v$ compatible with the transition $s(z, u, w) = z''$ is

$$s^a(z, u, z'') := \{w \in v : s(z, u, w) = z''\}. \quad (5.2)$$

In particular, when Σ is at a stable combination with the initial state $x_0 := x$ and the control input character u_0 , the adversarial input character w must satisfy

$$w \in v(x_0, u_0) := s^a(x_0, u_0, x_0) \cap v \subset v \quad (5.3)$$

Thus, $v(x_0, u_0)$ is the true initial adversarial uncertainty. For the transition from x_0 to x' to be possible, S must contain a path for each adversarial input character $w \in v(x_0, u_0)$, i.e., we must have $v(x_0, u_0) \subset \Pi_a S$. Otherwise, S would be incompatible with potential adversarial inputs. Further, let u_1 be a control input character, and define the set

$$S(x_0, u_0u_1) = \{\sigma \in S : \sigma|_1 = w|u_0u_1 \text{ for some } w \in B\}$$

of all strings of S whose control input starts with u_0u_1 . Clearly, u_1 can be a next control character only if it is compatible with all possible adversarial inputs, i.e., only if

$$v(x_0, u_0) \subset \Pi_a S(x_0, u_0u_1).$$

Also, the pair (x_0, u_1) must be detectable to facilitate fundamental mode operation of the closed loop machine, since the controller must react at the next stable state.

Now, let x_1 be the next stable state reached with the control input character u_1 ; the state x_1 can be read by the state feedback controller. The fact that Σ reached x_1 implies that the adversarial input value w must be within the set

$$v(x_0x_1, u_0u_1) := s^a(x_0, u_1, x_1) \cap v(x_0, u_0).$$

Continuing in this way, suppose that we are at step p of the path. Let $u_0u_1\dots u_p$ be the control input characters applied so far, and let $x_0x_1\dots x_p$ be the stable states Σ has passed as a result. The current uncertainty $v(x_0\dots x_p, u_0\dots u_p) \subset B$ about the adversarial input value is called the *residual adversarial uncertainty*. By iterating the earlier steps, we get

$$v(x_0\dots x_p, u_0\dots u_p) := s^a(x_{p-1}, u_p, x_p) \cap v(x_0\dots x_{p-1}, u_0\dots u_{p-1}). \quad (5.4)$$

Now, let $S(x_0x_1\dots x_p, u_0u_1\dots u_p)$ be the set of all strings $\sigma \in S$

having the control inputs $u_0u_1\dots u_p$ and taking Σ through the stable states $x_0, x_1, \dots, x_p$. For a control input character d , denote by $S(x_0x_1\dots x_p, u_0u_1\dots u_p d)$ the set of all strings $\sigma \in S(x_0x_1\dots x_p, u_0u_1\dots u_p)$ that have the character d as their next control input character. Then, the set of all adversarial input characters compatible with d is $\Pi_a S(x_0x_1\dots x_p, u_0u_1\dots u_p d)$. This set must be compatible with the residual adversarial uncertainty, namely,

(5.5) LEMMA. The character $d \in A$ can be used as the next control input character of the machine Σ only if $v(x_0x_1\dots x_p, u_0u_1\dots u_p) \subset \Pi_a S(x_0x_1\dots x_p, u_0u_1\dots u_p d)$. ♦

We show later that the condition of Lemma 5.5 is critical for the existence of a controller that automatically counteracts adversarial transitions. This leads us to the following.

(5.6) DEFINITION. Let $S \subset B|A^+$ be a set of strings taking Σ from a stable combination with the state x_0 to a stable combination with the state x' . Then, S is a *complete set* if the following hold for all $p = 0, 1, \dots$ and for all control input characters $d \in \Pi^{p+1} S(x_0x_1\dots x_p, u_0u_1\dots u_p)$:

- (i) $v(x_0x_1\dots x_p, u_0u_1\dots u_p) \subset \Pi_a S(x_0x_1\dots x_p, u_0u_1\dots u_p d)$, and
- (ii) The pair (x_p, d) is detectable with respect to the residual adversarial uncertainty $v(x_0x_1\dots x_p, u_0u_1\dots u_p)$. ♦

A complete set of strings can be replaced by one of bounded length, as follows. (For a set of strings $S \subset B|A^+$, denote by $|S|$ the maximal length of a control input string in S . For a finite set Z , denote by $\#Z$ the number of elements in Z .)

(5.7) LEMMA. Let Σ be in a stable combination at the state x_0 and the control input value u_0 . If there is a complete set of strings from x_0 to x' , then there also is such a complete set S satisfying $|S| \leq [\#v(x_0, u_0)](n-1)$.

Proof (sketch). Consider a string $\sigma = w|u = w|u_0u_1\dots u_k \in S$ and let $x_0x_1\dots x_k$ be the stable states through which Σ passes as a result of receiving the control input string u . Let v_i be the residual uncertainty at step i of the path, where $v_0 := v(x_0, u_0)$. Then, v_i is a monotone declining function of i , and its minimal value is not less than 1. Divide the interval $[0, k]$ into segments of constant residual uncertainty $[0, i_1]$, $[i_1+1, i_2]$, ..., $[i_m+1, k]$, where v_i is constant over each one of these subintervals. Since v_i is monotonously declining and its minimum cannot be less than 1, we get $m+1 \leq v(x_0, u_0)$, or $m \leq v(x_0, u_0) - 1$.

Now, if any of these subintervals $[i, i']$ has length $\ell \geq n$, then the string of states $x_i x_{i+1} \dots x_{i'}$ must contain a repeating state, say $x := x_p = x_r$, where $i \leq p < r \leq i + \ell$. Since $v_p = v_r$ by construction, the control input value u_p can be replaced by the control value u_r without disturbing the stable combination at step p . Then, steps $p+1, p+2, \dots, r$ can be eliminated from the string, yielding a new segment with the length of $\ell - (r - p)$. This process can be repeated again and again, until the length of the resulting segment is less than n . Applying the same procedure to each one of the segments, we obtain a new path of length not exceeding $(m+1)(n-1) = [\#v(x_0, u_0)](n-1)$. ♦

This brings us to the main result of this section.

(5.8) THEOREM. Let $\Sigma = (A \times B, X, f, v)$ be an asynchronous machine and let x and x' be two states of Σ . Then, the following two statements are equivalent.

- (i) There is a state feedback controller C that drives Σ from a stable combination with x to a stable combination with x' in fundamental mode operation.
- (ii) There is a complete set of strings $S \subset B|A^+$ taking Σ from a stable combination with x to a stable combination with x' .

Proof (sketch). Assume that (ii) is valid. We use S to build a state feedback controller $F(x, x', v)$ which, upon receiving the external input character $v \in A$, generates a string of control input characters that takes Σ from a stable combination with $x_0 := x$ to a stable combination with x' in fundamental mode operation. To this end, let Σ be in a stable combination with x_0 , and pick a control input character $u_1 \in \Pi^1 S$. Since S is a complete set of strings, (x_0, u_1) is detectable with respect to $v(x_0, u_0)$. Also, $v(x_0, u_0) \in \Pi_a S(x_0, u_0, u_1)$, so u_1 is compatible with all possible adversarial inputs. Denote by Ξ the state set of $F(x, x', v)$, by ϕ the recursion function of $F(x, x', v)$, and by η the output function of $F(x, x', v)$; let ξ_0 be the initial state of $F(x, x', v)$. We construct now ϕ and η .

Upon a detectable transition of Σ to x_0 with the control input character u_0 , the controller moves to a stable combination with its state ξ_1 , readying for controller action at the command v . To this end, set

$$\begin{aligned} \phi(\xi_0, (z, t)) &:= \xi_0 \text{ for all } (z, t) \neq (x_0, u_0), \\ \phi(\xi_0, (x_0, u_0)) &:= \xi_1, \phi(\xi_1, (x_0, u_0)) &:= \xi_1. \end{aligned}$$

While in the states ξ_0 or ξ_1 , the controller applies to the control input of Σ the external input it receives, namely

$$\eta(\xi_0, (z, t)) := t, \eta(\xi_1, (z, t)) := t \text{ for all } (z, t) \in X \times A,$$

If, while at ξ_1 , the controller $F(x, x', v)$ receives the external input character v (the command to start controller action), it moves to a stable combination with its state ξ_2 :

$$\begin{aligned} \phi(\xi_1, (z, t)) &:= \xi_1 \text{ for all } (z, t) \neq (x_0, v), \\ \phi(\xi_1, (x_0, v)) &:= \xi_2, \phi(\xi_2, (x_0, v)) &:= \xi_2. \end{aligned}$$

At ξ_2 , the $F(x, x', v)$ applies the first character u_1 of the control input string that takes Σ to the state x' , so we set

$$\eta(\xi_2, (x_0, t)) := u_1 \text{ for all } t \in A.$$

Since S is a complete set, u_1 makes Σ move to the state x_1 through a detectable transition. Whence, Σ is in a stable combination when it reaches x_1 . Upon detecting x_1 , the controller moves to a stable combination with its state ξ_3 :

$$\begin{aligned} \phi(\xi_2, (z, t)) &:= \xi_2 \text{ for all } (z, t) \neq (x_1, u_1), \\ \phi(\xi_2, (x_1, u_1)) &:= \xi_3, \phi(\xi_3, (x_1, u_1)) &:= \xi_3. \end{aligned}$$

At ξ_3 , the controller applies the next control input value $u_2 \in \Pi^2 S(x_0, x_1, u_0, u_1)$: $\eta(\xi_3, (x_1, t)) := u_2$ for all $t \in A$. Since S is a complete set of strings, the pair (x_1, u_2) is detectable for the

current adversarial uncertainty $v(x_0, x_1, u_0, u_1)$ and $v(x_0, x_1, u_0, u_1) \subset \Pi_a S(x_0, x_1, u_0, u_1, u_2)$. We continue in this way until the controller $F(x, x', v)$ generates the last input character of the string, bringing Σ to x' . By Lemma (5.7), the state x' can be reached in at most $(n-1)[\#v(x_0, u_0)]$ steps.

Conversely, assume that (i) is valid, and let $F(x, x', v)$ be the corresponding controller. Let $S \subset B|A^+$ be the set of strings that $F(x, x', v)$ may generate for the various possible adversarial input characters. To show that S is a complete set, consider a control input string $u_0 u_1 \dots u_p$ that $F(x, x', v)$ applies to Σ , and let $x_0 x_1 \dots x_p$ be the stable states through which Σ passes as a result. By (5.4), the adversarial uncertainty at step $p \geq 0$ is $v(x_0 x_1 \dots x_p, u_0 u_1 \dots u_p)$. By fundamental mode operation of the closed loop machine, the pair (x_p, u_{p+1}) is detectable with respect to $v(x_0 x_1 \dots x_p, u_0 u_1 \dots u_p)$. By Lemma 5.5, $v(x_0 x_1 \dots x_p, u_0 u_1 \dots u_p) \subset \Pi_a S(x_0 x_1 \dots x_p, u_0 u_1 \dots u_p, u_{p+1})$. Hence, S is a complete set. ♦

An algorithm for finding complete sets of strings is described in YANG and HAMMER [2007]. We turn now to an important definition. By (5.3), we have $v(x_0, u_0) \subset v$, so that $\#v(x_0, u_0) \leq \#v$. Invoking Lemma 5.7, we conclude that a complete set of strings S can always be selected so that

$$|S| \leq (n-1)(\#v). \quad (5.9)$$

Recalling the matrix $R(m, \Sigma, w)$, taking $m = (n-1)(\#v)$, and including all adversarial characters of v , we arrive at the following.

(5.10) DEFINITION. The $n \times n$ matrix

$$R(\Sigma, v) := v_{w \in v} R((n-1)(\#v), \Sigma, w)$$

is the *combined matrix of stable transitions* of an asynchronous machine Σ with adversarial uncertainty v . ♦

Considering (5.9), Lemma 5.7, and Theorem 5.8, we reach the following conclusion.

(5.11) COROLLARY. Let Σ be an asynchronous machine with adversarial uncertainty v and state set $X = \{x^1, \dots, x^n\}$. The statements below are equivalent for all $i, j = 1, 2, \dots, n$:

- (i) There is a state feedback controller that takes Σ from a stable combination with x^i to a stable combination with x^j in fundamental mode operation.
- (ii) The i, j entry of $R(\Sigma, v)$ includes a complete set of strings. ♦

6. COUNTERACTING ADVERSARIAL TRANSITIONS

Let Σ be an asynchronous machine at a stable combination (x, u, w) , when the adversarial input character switches to w' . An *adversarial transition* occurs if this switch causes Σ to move to a new stable state $x' \neq x$. In this section, we discuss state feedback controllers that automatically counteract adversarial transitions. In order to operate in fundamental mode, it must be possible for the controller to determine from the state of Σ whether or not it has reached the next stable state. The following is analogous to Definition 4.3.

(6.1) DEFINITION. Let Σ be at a stable combination with

the state x and the control input character u . The pair (x, u) is *adversarially detectable* if, after an adversarial transition, it can be determined from the current state of Σ whether or not Σ has reached its next stable state. ♦

Assume then that Σ is at a stable combination (x, u, w) , when the adversarial input character changes to w' , causing Σ to move to a stable combination with the state $x' \neq x$. This transition may consist of a number of intermediate steps, say $x_0 := x$, $x_1 := f(x_0, u, w')$, $x_2 := f(x_1, u, w')$, ..., $x_q := f(x_{q-1}, u, w') = x'$, $x_q := f(x_q, u, w')$. Similarly to (4.1) and (4.2), we denote

$$\begin{cases} \theta(x, u, w') := x_1 \dots x_q, \\ \theta[x, u, \varepsilon] := \{\theta(x, u, w') : w' \in \varepsilon\}. \end{cases} \quad (6.2)$$

The following has a proof similar to that of Theorem 4.4.

(6.3) THEOREM. The two statements are equivalent:

- (i) The pair (x, u) is adversarially detectable with respect to the adversarial uncertainty v .
- (ii) States of the set $s^v(x, u)$ appear only at the end of strings belonging to $\theta[x, u, v]$. ♦

To guarantee fundamental mode operation of the closed loop machine, the use of the machine Σ must be restricted to adversarially detectable pairs. This leads us to the following notion. (For a string $\sigma = w|u_1u_2\dots u_q \in B \times A^+$, denote by $\Pi_c^+ \sigma := u_q$ the last control input character of the string.)

(6.4) DEFINITION. Let Σ be an asynchronous machine with n states, adversarial uncertainty v , and combined matrix of stable transitions $R(\Sigma, v)$. The *reduced matrix of stable transitions* $R^r(\Sigma, v)$ is obtained by removing from each column $j = 1, 2, \dots, n$ of $R(\Sigma, v)$ all strings σ for which the pair $(x^j, \Pi_c^+ \sigma)$ is not adversarially detectable. ♦

(6.5) EXAMPLE. In Example 2.2, Σ has only one adversarial transition: $(x^1, b, \alpha) \rightarrow (x^1, b, \beta) \rightarrow (x^2, b, \beta)$. Then, $\theta[x^1, b, \alpha] = x^1$ and $\theta[x^1, b, \beta] = x^2$, so $\theta[x^1, b, v] = \{x^1, x^2\}$. Also, $s^v(x^1, u) = \{x^1, x^2\}$ here. Thus (x^1, b) is adversarially detectable by Theorem a381, and $R^r(\Sigma, v) = R(\Sigma, v)$. ♦

The set of adversarial input characters that give rise to an adversarial transition from a stable combination with the pair (x^s, u) to a stable combination with the pair (x^t, u) is

$$v(x^s, x^t, u) := s^a(x^s, u, x^t) \cap v. \quad (6.6)$$

Here, a transition occurs if and only if $v(x^s, x^t, u) \neq \emptyset$. We can state now the main result of this section; the proof is similar to that of Theorem 5.8.

(6.7) THEOREM. Let Σ be an asynchronous machine with the state set $\{x^1, x^2, \dots, x^n\}$ and the reduced matrix of stable transitions $R^r(\Sigma, v)$, and let x^s and x^t be states for which $v(x^s, x^t, u) \neq \emptyset$. Then, the following are equivalent:

- (i) There is a state feedback controller that automatically reverses an adversarial transition from the state x^s to the state x^t in fundamental mode operation.
- (ii) The entry $R_{ts}^r(\Sigma, v)$ includes a complete set of strings with respect to the adversarial uncertainty $v(x^s, x^t, u)$. ♦

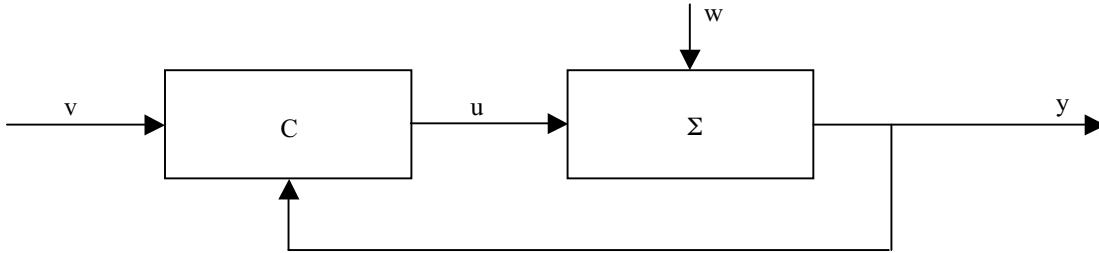
7. REFERENCES

- BARRETT, G. and LAFORTUNE, S. [1998], "Bisimulation, the supervisory control problem, and strong model matching for finite state machines," *Discrete Event Syst.: Theory Appl.*, **vol. 8**, no. 4, pp. 377–429.
- DIBENEDETTO, M.D., SALDANHA, A. and SANGIOVANNI-VINCENTELLI, A. [1994], "Model matching for finite state machines," *Proc. IEEE Conf. on Decision and Control*, **vol. 3**, pp. 3117–3124.
- GENG, X.J. and HAMMER J. [2004], "Asynchronous sequential machines: input/output control", *Proc. 12th Mediterranean Conf. on Control and Automation*, Kusadasi, Turkey, June 2004. [2005], "Input/output control of asynchronous sequential machines," *IEEE Trans. Automat. Contr.*, **vol. 50**, no. 12, pp. 1956–1970.
- HAMMER, J. [1994], "On some control problems in molecular biology," *Proc. IEEE Conf. on Decision and Control*, 1994. [1995], "On the modeling and control of biological signal chains," *Proc. IEEE Conf. on Decision and Control*, 1995. [1996a], "On the corrective control of sequential machines," *Int. J. Contr.*, **vol. 65**, no. 65, pp. 249–276. [1996b], "On the control of incompletely described sequential machines," *Int. J. Contr.*, **vol. 63**, no. 6, pp. 1005–1028. [1997], "On the control of sequential machines with disturbances," *Int. J. Contr.*, **vol. 67**, no. 3, pp. 307–331.
- KOHAVALI, Z., [1970], *Switching and Finite Automata Theory*, McGraw-Hill Book Company, New York.
- MURPHY, T.E., GENG, X.J. and HAMMER, J., [2002], "Controlling races in asynchronous sequential machines," *Proc. IFAC World Congress*, Barcelona, Spain, 2002. [2003], "On the control of asynchronous machines with races," *IEEE Trans. Automat. Contr.*, **vol. 48**, no. 6, pp. 1073–1081.
- RAMADGE, P.J.G. and WONHAM, W.M., [1987], "Supervisory control of a class of discrete event processes," *SIAM J. Contr. Optim.*, **vol. 25**, no. 1, pp. 206–230.
- THISTLE, J.G. and WONHAM, W.M., [1994], "Control of infinite behavior of finite automata," *SIAM J. Contr. Optim.*, **vol. 32**, no. 4, pp. 1075–1097.
- VENKATRAMAN, N. and HAMMER, J., [2006a], "Stable realizations of asynchronous sequential machines with infinite cycles," *Proc. 2006 Asian Control Conf.*, Bali, Indonesia. [2006b], "Controllers for asynchronous machines with infinite cycles," *Proc. 17th Int. Symp. on Mathematical Theory of Networks and Systems*, Kyoto, Japan. [2006c], "On the control of asynchronous sequential machines with infinite cycles," *Int. J. Contr.*, **vol. 79**, no. 7, pp. 764–785.
- YANG J.M. and HAMMER, J., [2007], "State feedback control of asynchronous sequential machines with adversarial inputs," in preparation.

1. INTRODUCTION

Unauthorized and adversarial input agents have, regretfully, become an unavoidable feature of modern day computing. These agents aim to interfere with the proper operation of computing systems, often attempting to subdue them to hostile objectives. The present paper addresses the question of how a computing system can be made immune to such attempts to interfere with its operation. The approach taken here is based on automatic control. Specifically, we propose to deploy automatic controllers that continually monitor a computing system and take corrective action whenever an adversarial input attempts to intercede. The paper presents necessary and sufficient conditions for the existence of such controllers; when a controller exists, an algorithm for its design is also provided.

In formal terms, the discussion revolves around asynchronous sequential machines that have two input signals: one input signal facilitates control of the machine, while the other input signal is used by an adversarial agent (or by a disturbance) to interfere with the operation of the machine. The control diagram is as follows.



(1.1)

Here, Σ is the asynchronous sequential machine being controlled, and C is another asynchronous sequential machine that serves as a controller. The machine Σ has two inputs: u is the *control input* used to steer the machine to proper operation; and w is the *adversarial input* operated by an adversarial agent or by a disturbance. The purpose of the controller C is to counteract action at the input w and to endow the closed loop machine with desirable behavior. Section 7 presents necessary and sufficient conditions for the existence of such a controller C , and section 8 describes the structure of the controller. The closed loop machine described by the diagram is denoted by $\Sigma_c(v, w)$, where v is the external input of the closed loop machine. In our present discussion, C is a state feedback controller - it has access to the state of the machine Σ .

Our discussion is within the context of model matching. Let Σ' be an asynchronous sequential machine that describes the desired behavior of the closed loop system. We refer to Σ' as the *model*. Of course, the model Σ' is not affected by the adversarial input w - it has no adversarial input; Σ' accepts only user commands as its input. The purpose of the controller C is to drive the machine Σ so that, from a user's perspective, the closed loop system matches the behavior of the model Σ' , irrespective of actions taken at the adversarial input. In other words, we would like to have $\Sigma_c(\bullet, w) = \Sigma'(\bullet)$ for all adversarial input values w . This is the *perturbed model matching problem*.

Recall that an asynchronous sequential machine has two kinds of states : *stable states*, namely, states at which the machine dwells indefinitely with its present input value, and *transient states* - states the machine passes transiently along its way to the next stable state. Only stable states are perceivable by the machine's user; transient states are traversed by the machine very quickly (ideally, in zero time), and are imperceptible to the user. Thus, when eliminating the effects of an adversarial input, it is only necessary to eliminate the effects on stable states; effects on transient states are unnoticeable and, therefore, inconsequential.

The ability to control an asynchronous machine Σ so as to eliminate the effects of adversarial inputs and match a specified model depends on certain features of reachability and detectability introduced in section 4. The essence of these features is condensed in sections 6 and 7 into a numerical matrix of zeros and ones, called a *skeleton matrix*. The skeleton matrix characterizes the possibilities of controlling the machine Σ in the presence of an adversarial input. More specifically, the skeleton matrix characterizes those aspects of the performance of Σ that can be preserved by an automatic controller despite activity at the adversarial input. In section 7, we show that the

perturbed model matching problem is solvable if and only if the skeleton matrix satisfies a certain numerical inequality (compare to MURPHY, GENG, and HAMMER [2002 and 2003]).

The present paper deals with the control of asynchronous sequential machines utilizing the formalism of MURPHY, GENG, and HAMMER [2002 and 2003], GENG and HAMMER [2004 and 2005], and VENKATRAMAN and HAMMER [2006a, 2006b, and 2006c]. Of course, asynchronous sequential machines are a topic within the general area of discrete mathematics. To a large extent, our terminology and notation follow EILENBERG [1974]. Studies dealing with other aspects of the control of discrete event systems can be found in RAMADGE and WONHAM [1987], HAMMER [1994, 1995, 1996a, 1996b, 1997], DIBENEDETTO, SALDANHA, and SANGIOVANNI-VINCENTELLI [1994], THISTLE and WONHAM [1994], BARRETT and LAFORTUNE [1998], the references cited in these papers, and others. It seems, however, that these publications do not address issues that are peculiar to the function of asynchronous machines, such as the avoidance of critical races and the distinction between stable states and transient states.

An important aspect of the operation of asynchronous sequential machines is fundamental mode operation (e.g., KOHAVI [1970]). Under fundamental mode operation, the input variables of an asynchronous machine are kept constant while the machine undergoes state transitions. Fundamental mode operation comes to guarantee deterministic behavior. Indeed, if an input value is changed while a machine undergoes state transitions, then the state at which the input change occurs becomes unpredictable; this may result in an unpredictable outcome.

In the case of configuration (1.1), fundamental mode operation means that (i) the controller C must be in a stable state while Σ undergoes transitions, and (ii) the machine Σ must be in a stable state while C undergoes transitions. All systems considered in this paper are designed to operate in fundamental mode.

The paper is organized as follows. Section 2 reviews and expands the basic notation and framework of our discussion. Section 3 introduces adversarial inputs and examines some of their potential effects. Two notions that are critical to the solution of the adversarial model matching problem - the notions of reachability and detectability - are discussed in section 4. Section 5 introduces a test that determines whether or not an adversarial action can be counteracted. Necessary and sufficient conditions for the existence of a controller that solves the perturbed model matching problem are derived in sections 6 and 7, while the structure of the controller is described in section 8. The paper concludes in section 9 with a comprehensive example.

2. NOTATION AND BASICS

Let A be a finite non-empty alphabet, let A^* be the set of all finite strings of characters of A , and let A^+ be the set of all non-empty strings in A^* . To simplify our notation later, we assume that the alphabet A does not include the digits 0 and 1. 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 $w := w_1 w_2$ obtained by appending w_2 to the end of w_1 . A partial function $f : S_1 \rightarrow S_2$ is a function whose domain is a subset of S_1 .

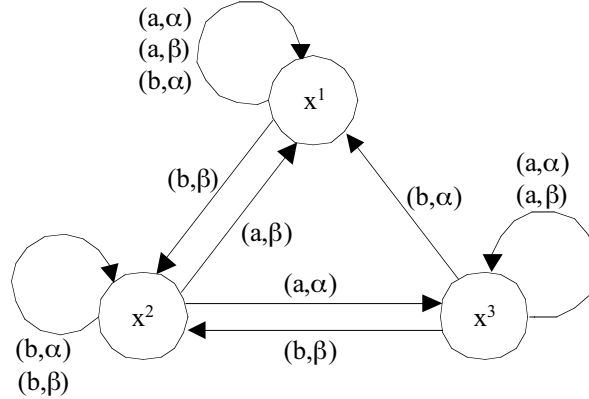
Consider an asynchronous sequential machine with two inputs: a control input through which the machine is operated and an adversarial input that attempts to interfere with the operation of the machine. We represent such a machine by a sextuple $\Sigma = (A \times B, Y, X, x_0, f, h)$, where A is the control input alphabet, B is the adversarial input alphabet, Y is the output alphabet, X is a set of n states, and x_0 is the initial state of the machine; $f : X \times A \times B \rightarrow X$ (the *recursion function*) and $h : X \rightarrow Y$ (the *output function*) are partial functions. The machine Σ operates recursively according to

$$(2.1) \quad \begin{aligned} x_{k+1} &= f(x_k, u_k, w_k), \\ y_k &= h(x_k), \quad k = 0, 1, 2, \dots \end{aligned}$$

Here, $u_0, u_1, u_2, \dots$ is the *control input sequence*, while $w_0, w_1, w_2, \dots$ is the *adversarial input sequence*; $x_0, x_1, x_2, \dots$ is the sequence of the states through which the machine passes, and $y_0, y_1, y_2, \dots$ is the sequence of output values. The integer k represents the *step counter*; it advances by one upon any change of the machine's inputs or state. Having selected the Moore representation for Σ , the output function h does not depend on the input variables u and w (e.g., KOHAVI [1970]).

(2.2) EXAMPLE. The following represents an asynchronous machine Σ with adversarial input. Here, the control input alphabet is $A = \{a, b\}$; the adversarial input alphabet is $B = \{\alpha, \beta\}$; the state set is $X = \{x^1, x^2, x^3\}$. We assume here that the state of the machine is also its output, so that the output function h is the identity function. The recursion function f of Σ is described by the following diagram:

Σ :



Alternatively, the transition function f can be characterized by the following transition table.

	(a,α)	(a,β)	(b,α)	(b,β)
x^1	x^1	x^1	x^1	x^2
x^2	x^3	x^1	x^2	x^2
x^3	x^3	x^3	x^1	x^2

A triplet (x, u, w) is a *valid combination* of Σ if it is included in the domain of the function f . Occasionally, we use a single character for the two inputs, as in $\alpha_k = (u_k, w_k)$, $k = 0, 1, 2, \dots$. The input sequence $\alpha_0, \alpha_1, \alpha_2, \dots$ is *permissible* if all pairs $(x_0, \alpha_0), (x_1, \alpha_1), (x_2, \alpha_2), \dots$ are valid.

Consider an input string with repeated characters, say *aaabbbcccc*. In *compressed notation*, the repetitions are omitted, so our string becomes simply *abc*. This notation conforms to the way the string is applied in practice - the first input character is kept constant for the first three steps, then the second character is kept constant for two steps, and, finally, the last input character is kept constant for four steps.

The machine Σ is an *input/state machine* when its output is the state, namely, when $Y = X$ and $h(x) = x$ for all $x \in X$. Then, $y_k = x_k$ for all $k = 0, 1, 2, \dots$, and the machine is described by the recursion

$$(2.3) \quad \Sigma : x_{k+1} = f(x_k, u_k, w_k).$$

An input/state machine is represented by a quadruple $\Sigma = (A \times B, X, x_0, f)$. The present paper deals with the control of asynchronous input/state machines.

A valid triplet (x, u, w) is a *stable combination* if $x = f(x, u, w)$, namely, if the state x is a fixed point of the recursion function f . An asynchronous machine lingers at a stable combination until a change occurs at one of its inputs. A triplet (x, u, w) that is not a stable combination is a *transient combination*.

A transient triplet (x, u, w) initiates a chain of transitions $x_1 = f(x, u, w)$, $x_2 = f(x_1, u, w)$, $\dots$, where the input characters u and w are kept fixed while the states change. This chain of transitions may or may not terminate. If it terminates, then there is an integer $q \geq 1$ for which $x_q = f(x_q, u, w)$, i.e., (x_q, u, w) is a stable combination. Then, x_q is the *next stable state* of x with the input pair (u, w) . If the chain of transitions does not terminate, then the triplet (x, u, w) is part of an *infinite cycle*. In this paper, we restrict our attention to machines that have no infinite cycles. Thus, in our case, every valid triple (x, u, w) has a next stable state. For future reference, it is convenient to record this fact.

(2.4) LEMMA. In an asynchronous machine without infinite cycles, every valid combination has a next stable state.

When operating an asynchronous machine, one has to be careful to prevent situations where two or more variables change value at the same time. A simultaneous change of two or more variables may cause an asynchronous machine to become unpredictable (e.g., KOHAVI [1970]). Thus, it is common to enforce a policy where only one variable is allowed to change value at any instant of time. When this policy is enforced, the machine Σ operates in *fundamental mode*. In practice, almost all asynchronous machines are operated in fundamental mode.

The asynchronous machine Σ of (2.3) involves three variables: the state x and the input variables u and w . In fundamental mode operation, not more than one of these variables can change value at any instant of time. This leads to the following.

(2.5) DEFINITION. Let $\Sigma = (A \times B, X, f)$ be an asynchronous input/state machine with the two input variables u and w . The machine Σ operates in *fundamental mode* if u and w change values only when Σ is in a stable combination, and then at most one at a time. ♦

All the machines discussed in this paper operate in fundamental mode. Note that fundamental mode operation is not an overly restrictive requirement. An asynchronous machine reaches its next stable combination very quickly - ideally, in zero time; thus, it is rather unlikely that changes in u or in w will occur while Σ is in transition. Similarly, the probability that the two independent variables u and w will change values simultaneously is zero, since the two are not synchronized. Fundamental mode operation is most common in practice, and we adopt it as our mode of operation.

Transitions from one stable combination of Σ to another are governed by its *stable recursion function* s , which is defined as follows. For a valid triplet (x, u, w) of the machine Σ , let x' be the next stable state; the stable recursion function $s : X \times A \times B \rightarrow X$ is defined by the assignment $s(x, u, w) := x'$. As Σ has no infinite cycles, every valid triplet has a next stable state, and hence s is defined on all valid triplets. The stable recursion function s induces the *stable state machine* Σ_s of Σ , which is an asynchronous input/state machine given by $(A \times B, X, s)$. Here, the stable recursion function s replaces the recursion function f of Σ .

Consider an input string $\alpha := \alpha_0 \alpha_1 \dots \alpha_{m-1}$, where $\alpha_i \in A \times B$. Writing in terms of components, we have that $\alpha_i = (u_i, w_i)$, with $u_i \in A$ and $w_i \in B$, $i = 0, \dots, m-1$. Assume that Σ is in a stable combination at the initial state x_0 , when the input string α is applied. In fundamental mode operation, the first input value α_0 must remain fixed until Σ reaches its next stable state $x_1 := s(x_0, \alpha_0)$ (ideally, this transition is completed in zero time). Anytime thereafter, one of the input variables u or w can change, providing the next input pair α_1 . These input values remain fixed until the next stable state $x_2 := s(s(x_0, \alpha_0), \alpha_1)$ is reached. This process continues until the last stable state $x_m := s(\dots s(s(x_0, \alpha_0), \alpha_1), \alpha_2) \dots, \alpha_{m-1})$ is reached. It is convenient to use the shorthand notation

$$(2.6) \quad s(x_0, \alpha) = s(\dots s(s(s(x_0, \alpha_0), \alpha_1), \alpha_2) \dots, \alpha_{m-1}), \alpha \in (A \times B)^+.$$

When the machine $\Sigma = (A \times B, X, f)$ is connected in the closed loop configuration (1.1), the output of the controller C serves as the control input of Σ . We therefore use A as the output alphabet of C . Also, the input variable v of the controller C in the diagram is really intended to control the operation of Σ , so we use A as the alphabet of v as well. In addition to v , the controller C accepts the state X of Σ as another input, so the input set of C is $A \times X$. Letting Ξ be the state set, ϕ the recursion function, and η the output function of the controller, we can write $C = (A \times X, A, \Xi, \xi_0, \phi, \eta)$, where ξ_0 is the initial state of C .

In fundamental mode operation of the configuration (1.1), only one of the asynchronous machines Σ and C can undergo transitions at any instant of time. Adapting GENG and HAMMER [2004 Proposition 1.4] to our present case, we obtain the following.

(2.7) PROPOSITION. Let $\Sigma = (A \times B, X, f)$ and $C = (A \times X, A, \Xi, \xi_0, \phi, \eta)$ be asynchronous machines interconnected in the configuration (1.1). Then, the configuration operates in fundamental mode if and only if all the following hold:

- (i) C is in a stable combination while Σ undergoes transitions, and Σ is in a stable combination while C undergoes transitions.
- (ii) The inputs u , w , and v change only while Σ and C are in a stable combination, and then only one at a time. ♦

In view of Proposition 2.7, the controller C must be designed so that it commences transitions only after verifying that Σ has reached a stable combination. Similarly, C must adopt a stable combination immediately prior

to inducing a change at the input of Σ . The controller C designed in section 8 below satisfies these requirements. Fundamental mode operation assures that all transitions of the composite system are unambiguous and deterministic.

3. ADVERSARIAL INPUTS AND MODEL MATCHING

Our next objective is to discuss the implications of the adversarial input on the machine $\Sigma = (A \times B, X, x_0, f)$ of (2.3). The critical issue is, of course, the fact that the adversarial input character w_k is, in general, not specified. The only a-priori information available about w_k is that it belongs to a specified subset $v \subset B$ called the *adversarial uncertainty*. Adding the adversarial uncertainty v to the description of Σ , we obtain the quintuple $\Sigma = (A \times B, X, x_0, f, v)$.

When the adversarial uncertainty v includes more than one character, the precise value of the adversarial input is not specified; this entails that the state of the machine Σ may not be known. Indeed, starting from the initial state x_0 and applying the initial control input value u_0 , the state of Σ after the first step can be any member of the set

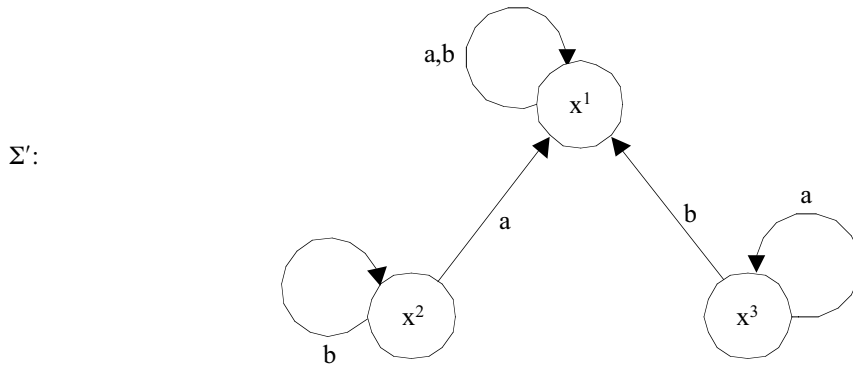
$$f[x_0 \times u_0 \times v] := \bigcup_{w \in v} f(x_0, u_0, w) \subset X.$$

When this set contains more than one state, one cannot predict the precise state of Σ after the first step. In such case, we are faced with the control of an asynchronous machine in the presence of uncertainty.

Recall that two asynchronous machines Σ_1 and Σ_2 are *stably equivalent* if the stable state machines Σ_{1s} and Σ_{2s} are equivalent (see GENG and HAMMER [2004], [2005] for more details). Stably equivalent machines have equivalent functionality and are identical from a user's point of view. We are now ready to formally formulate the main topic of our discussion.

(3.1) THE PERTURBED MODEL MATCHING PROBLEM. Let $\Sigma = (A \times B, X, f, v)$ be an input/state asynchronous machine with an adversarial input, and let $\Sigma' = (A, X, s')$ be a stable-state input/state machine with no adversarial input. Find necessary and sufficient conditions for the existence of a controller C for which the closed loop machine $\Sigma_c(v, w)$ is stably equivalent to $\Sigma'(v)$ for all $w \in v$ and all $v \in A$. If such a controller exists, derive an algorithm for its design. ♦

(3.2) EXAMPLE. A model machine Σ' . This input/state machine has the control input alphabet $A = \{a, b\}$ and the state set $X = \{x^1, x^2, x^3\}$. It is a stable state machine with the stable transition function s' given by the following transition diagram; the model has no adversarial input.



Alternatively, the stable transition function s' of Σ' can be described by the following table of transitions.

	a	b
x^1	x^1	x^1
x^2	x^1	x^2
x^3	x^3	x^1

◆

Now, let s be the stable recursion function of the asynchronous machine $\Sigma = (A \times B, X, f, v)$. Assume that Σ is in a stable combination with the state x when the control input takes the value u . As Σ has no infinite cycles, it follows by Lemma 2.4 that there is a next stable state x' of Σ . The exact value of x' is usually impossible to predict, since the value w of the adversarial input is not known. All possible values of x' are given by the set

$$(3.3) \quad s^v(x, u) := s[x, u, v] = \{s(x, u, w) : w \in v\} \subset X.$$

Letting $P(X)$ be the family of all subsets of the state set X of Σ , we thus obtain the function $s^v : X \times A \rightarrow P(X)$ called the *perturbed stable recursion function* of Σ .

4. DETECTABILITY AND REACHABILITY

4.1. Detectability

As we have seen in Proposition 2.7, fundamental mode operation of the closed loop (1.1) requires the controller C to remain in a stable combination until the machine Σ has reached its next stable state. Consequently, it must be possible for C to determine whether or not Σ has reached its next stable state. We examine now the conditions under which the latter is possible.

To this end, let v be the adversarial uncertainty of the machine Σ , and let $w \in v$ be the adversarial input character. Assume that Σ is in a stable combination with the state x , when the control input variable takes the value u . The machine Σ embarks then on a chain of state transitions given by

$$(4.1) \quad \theta(x, u, w) := \{x_1 := f(x, u, w), x_2 := f(x_1, u, w), \dots, x_{i(u, w)} := f(x_{i(u, w)-1}, u, w)\},$$

where $x_{i(u, w)}$ is the stable state reached at the end. Clearly, the number of steps $i(u, w)$ and the state $x_{i(u, w)}$ depend on the adversarial input character $w \in v$. According to (3.3), we must have $x_{i(u, w)} \in s^v(x, u)$. The set

$$(4.2) \quad \theta[x, u, v] := \{\theta(x, u, w) : w \in v\}$$

includes all state strings that form chains of transitions consistent with the adversarial uncertainty v , given that Σ starts from the state x with the control input character u . The following statement shows that a string in $\theta[x, u, v]$ includes no repeating states.

(4.3) LEMMA. Let $\Sigma = (A \times B, X, f, v)$ be an asynchronous machine with no infinite cycles. Let x be a state of Σ and let u be a control input value. Then, a string $\theta \in \theta[x, u, v]$ includes no repeating states.

Proof. Consider a string $\theta = \{x_1, x_2, \dots, x_{i(u, w)}\} \in \theta[x, u, v]$. First, if $i(u, w) = 1$, then θ includes only one state, and clearly has no repeating states. Consider then the case where $i(u, w) > 1$. Assume, by contradiction, that $x_i = x_j$ for some integers $1 \leq i < j \leq i(u, w)$. Now, if $j = i(u, w)$, then the triple $(x_i, u, w) = (x_j, u, w) = (x_{i(u, w)}, u, w)$ is a stable combination. If view of (4.1), this implies that θ terminates at step i , so that $i = j = i(u, w)$, a contradiction. Next, consider the case where $j < i(u, w)$. Using the recursion function of Σ , we get that $x_{i+1} = f(x_i, u, w) = f(x_j, u, w) = x_{j+1}$, $x_{i+2} = f(x_{i+1}, u, w) = f(x_{j+1}, u, w) = x_{j+2}$, ..., which implies that the machine Σ has an infinite cycle $x_i, x_{i+1}, x_{i+2}, \dots, x_j, x_{j+1}, x_{j+2}, \dots$ contradicting our assumption. This concludes our proof. ◆

The following notion underlies the controller's ability to determine whether or not the controlled machine Σ has reached its next stable state.

(4.4) DEFINITION. Let $\Sigma = (A \times B, X, f, v)$ be an input/state asynchronous machine, let $x \in X$ be a state of Σ , and

let $u \in A$ be a control input character. Assume that Σ is in a stable combination with the state x , when the control input character changes to u . Then, the pair (x, u) is *detectable* if it is possible to determine from the state of Σ whether or not Σ has reached its next stable combination. ♦

The following is a test that allows us to ascertain whether a given pair (x, u) is detectable.

(4.5) THEOREM. Let $\Sigma = (A \times B, X, f)$ be an input/state asynchronous machine with an adversarial input, and let ε be the adversarial uncertainty. Let $x \in X$ be a state of Σ , let $u \in A$ be a control input character, and assume that Σ is in a stable combination with the state x when the control input character changes to u . Then, in the notation of (3.3) and (4.2), the following two statements are equivalent:

- (i) The pair (x, u) is detectable.
- (ii) States of the set $s^\varepsilon(x, u)$ appear only at the end of strings belonging to $\theta[x, u, \varepsilon]$.

Proof. We use the notation of (4.1). Let $w \in \varepsilon$ be the active adversarial input character, and let $\theta(x, u, w) = \{x_0, x_1, x_2, \dots, x_{i(u, w)}\}$ be the string of transitions initiated when the control input character switches to u . Assume that there is an integer $0 \leq j < i(u, w)$ for which $x_j \in s^\varepsilon(x, u)$. Since $j < i(u, w)$, it follows that (x_j, u, w) is a transient combination, as the chain of transitions continues after j . By the definition of the set $s^\varepsilon(x, u)$, the inclusion $x_j \in s^\varepsilon(x, u)$ implies that there is an adversarial input character $w' \in \varepsilon$ such that the triple (x_j, u, w') is a stable combination. Summarizing, the state x_j forms a transient combination in the triple (x_j, u, w) while forming a stable combination in the triple (x_j, u, w') . Consequently, when Σ is in the state x_j , one cannot tell whether Σ is in a stable combination or not. In other words, when (ii) is invalid, so is (i).

Conversely, assume that a state $x' \in s^\varepsilon(x, u)$ can appear only at the end of a string belonging to $\theta(x, u, \varepsilon)$. Then, by the definition of $\theta(x, u, \varepsilon)$, the machine Σ is in a stable combination if and only if it is in a state $x' \in s^\varepsilon(x, u)$. In other words, one can determine from the state of Σ whether or not it has reached its next stable combination, and (x, u) is detectable. Thus, (ii) implies (i), and our proof concludes. ♦

Thus, for a detectable pair (x, u) , a state feedback controller can determine whether or not Σ has reached its next stable state simply by checking whether the current state of Σ is a member of $s^\varepsilon(x, u)$. If it is, then Σ has reached its next stable state; if it is not, then Σ has not yet reached its next stable state. For pairs that are not detectable, it is not possible to determine from the current state whether Σ has reached its next stable combination.

(4.6) REMARK. Here, we assume that the state feedback controller provides information only about the current state of the machine Σ , and does not keep track of the entire trajectory of states traversed by Σ on its way to the current state. Controllers that keep track of the entire trajectory of Σ (i.e., controllers that record the "burst" of Σ) have a more complex structure and will be discussed in a separate report. ♦

4.2. Reachability

We turn now to an examination of the reachability features of asynchronous machines with adversarial input. First, we adapt to our present setting the following notion from MURPHY, GENG, and HAMMER [2002] and [2003].

(4.7) DEFINITION. Let $\Sigma = (A \times B, X, f, v)$ be an input/state asynchronous machine with adversarial input, having the state set $X = \{x^1, x^2, \dots, x^n\}$. Let s be the stable transition function of Σ , let $u \in A$ be a control input character, and let $w \in B$ be an adversarial input character.

(i) The *one-step sample control transition matrix* $\gamma(\Sigma, w)$ of Σ is an $n \times n$ matrix whose (i, j) entry $\gamma_{ij}(\Sigma, w)$ consists of all control input characters $u \in A$ for which $x^j = s(x^i, u, w)$; if there are no such control input characters, then $\gamma_{ij}(\Sigma, w) := N$, where N is a character not in A or in B .

(ii) The *one-step sample adversarial transition matrix* $\lambda(\Sigma, u)$ of Σ is an $n \times n$ matrix whose (i, j) entry $\lambda_{ij}(\Sigma, u)$ consists of all adversarial input characters $w \in B$ for which $x^j = s(x^i, u, w)$; if there are no such adversarial input characters, then $\lambda_{ij}(\Sigma, u) := N$, where N is a character not in A or in B . ♦

More formally, we can write

$$\gamma_{ij}(\Sigma, w) = \begin{cases} N & \text{if } \{u \in A : x^j = s(x^i, u, w)\} = \emptyset, \\ \{u \in A : x^j = s(x^i, u, w)\} & \text{otherwise,} \end{cases}$$

and

$$\lambda_{ij}(\Sigma, u) = \begin{cases} N & \text{if } \{w \in V : x^j = s(x^i, u, w)\} = \emptyset, \\ \{w \in V : x^j = s(x^i, u, w)\} & \text{otherwise,} \end{cases}$$

$i, j = 1, \dots, n$.

In order to keep an explicit record of the adversarial input character, we introduce the *one-step matrix of stable transitions* $\rho(\Sigma, w)$, where we add the prefix $w|$ to all entries of $\gamma(\Sigma, w)$, namely,

$$\rho_{ij}(\Sigma, w) := \{w|v : v \in \gamma_{ij}(\Sigma, w)\}, i, j = 1, 2, \dots, n.$$

(4.8) EXAMPLE. Consider the machine Σ of Example 2.2. The one-step matrices of stable transitions of Σ are

$$\rho(\Sigma, \alpha) = \begin{pmatrix} \{\alpha|a, \alpha|b\} & \{\alpha|N\} & \{\alpha|N\} \\ \{\alpha|N\} & \{\alpha|b\} & \{\alpha|a\} \\ \{\alpha|b\} & \{\alpha|N\} & \{\alpha|a\} \end{pmatrix}, \rho(\Sigma, \beta) = \begin{pmatrix} \{\beta|a\} & \{\beta|b\} & \{\beta|N\} \\ \{\beta|a\} & \{\beta|b\} & \{\beta|N\} \\ \{\beta|N\} & \{\beta|b\} & \{\beta|a\} \end{pmatrix} \blacklozenge$$

To work with the entries of $\rho(\Sigma, w)$, we define the projections $\Pi_a : B|(A^+ \cup N) \rightarrow B$ (projection onto the adversarial input value) and $\Pi_c : B|(A^+ \cup N) \rightarrow (A^+ \cup N)$ (projection onto the control input value) by setting

$$\Pi_a w|u := \begin{cases} w & \text{if } u \neq N, \\ \emptyset & \text{else,} \end{cases}$$

and

$$\Pi_c w|u := u \text{ for all } w|u \in B|(A^+ \cup N).$$

Next, given two sets of strings $s_1, s_2 \subset B|(A^+ \cup N)$, we define an operation $s_1 \vee s_2$ that is akin to the union of the two sets, with N being handled like the empty set it represents. Specifically, we delete from the union all elements $w|N$ for which w also appears with a nonempty control input string. Using $\setminus$ to denote the difference set, we have

$$s_1 \vee s_2 := [s_1 \cup s_2] \setminus s_N,$$

where s_N consists of all elements $w|N \in s_1 \cup s_2$ for which $[w|A^+] \cap [s_1 \cup s_2] \neq \emptyset$.

Recall that concatenation is an operation that combines two strings into one longer string. When dealing with pairs of strings of the form $w|u \in B|(A^+ \cup N)$, concatenation operates only on strings with the same adversarial input character, concatenating the control inputs and leaving the adversarial input character unchanged. Explicitly, given two strings $w_1|u_1, w_2|u_2 \in B|(A^+ \cup N)$, set

$$\text{conc}(w_1|u_1, w_2|u_2) := \begin{cases} w_1|u_1u_2 & \text{if } w_1 = w_2 \text{ and both } u_1, u_2 \neq N, \\ w_1|N, w_2|N & \text{otherwise.} \end{cases}$$

For two subsets of strings $\sigma_1, \sigma_2 \subset B|(A^+ \cup N)$, the concatenation is defined by

$$\text{conc}(\sigma_1, \sigma_2) := \vee_{s_1 \in \sigma_1, s_2 \in \sigma_2} \text{conc}(s_1, s_2).$$

Further, we define an operation that resembles matrix multiplication. Let P and Q be two $n \times n$ matrices with entries in the set $B|(A^+ \cup N)$. The *combination* PQ is an $n \times n$ matrix with the entries

$$(PQ)_{ij} := \vee_{k=1, \dots, n} \text{conc}(P_{ik}, Q_{kj}), i, j = 1, 2, \dots, n.$$

Using this operation, we raise the one step matrix of stable transitions to the power of k , to get the matrix

$$\rho^k(\Sigma, w) = \rho^{k-1}(\Sigma, w) \rho(\Sigma, w), k = 1, 2, \dots$$

By construction, the i, j entry of $\rho^k(\Sigma, w)$ consists of all strings of the form $w|u$ that take the machine Σ from a stable combination with the state x^i to a stable combination with the state x^j in exactly k steps; if no such transition is possible, then $u = N$.

(4.9) EXAMPLE. Continuing from Example 4.8, we obtain (omitting repeated characters):

$$\rho^4(\Sigma, \alpha) = \begin{pmatrix} \{\alpha|a, \alpha|aba, \alpha|baba, \alpha|ba, \alpha|abab, \alpha|ab, \alpha|bab, \alpha|b\} & \{\alpha|N\} & \{\alpha|N\} \\ \{\alpha|aba, \alpha|baba, \alpha|abab, \alpha|bab, \alpha|ab\} & \{\alpha|b\} & \{\alpha|ba, \alpha|a\} \\ \{\alpha|baba, \alpha|ba, \alpha|aba, \alpha|abab, \alpha|bab, \alpha|b, \alpha|ab\} & \{\alpha|N\} & \{\alpha|a\} \end{pmatrix}$$

$$\rho^4(\Sigma, \beta) = \begin{pmatrix} \{\beta|a, \beta|baba, \beta|aba, \beta|ba\} & \{\beta|abab, \beta|bab, \beta|ab, \beta|b\} & \{\beta|N\} \\ \{\beta|a, \beta|baba, \beta|aba, \beta|ba\} & \{\beta|abab, \beta|bab, \beta|ab, \beta|b\} & \{\beta|N\} \\ \{\beta|baba, \beta|ba, \beta|aba\} & \{\beta|abab, \beta|bab, \beta|b, \beta|ab, \beta|a\} & \{\beta|a\} \end{pmatrix} \blacklozenge$$

The matrix

$$(4.10) \quad R(m, \Sigma, w) := v_{i=1, \dots, m} \rho^i(\Sigma, w)$$

is the *matrix of m stable transitions* of Σ . It characterizes all the transitions of Σ that can be accomplished in m or fewer stable steps. When we allow m to grow indefinitely, we obtain the *extended matrix of stable transitions*

$$R^*(\Sigma, w) := v_{i \geq 1} \rho^i(\Sigma, w),$$

which characterizes all stable transitions of the machine Σ .

Using the fact that Σ has only n states, an argument similar to the one employed in MURPHY, GENG, and HAMMER [2003, Proposition 3.9], yields the following.

(4.11) LEMMA. Let $\Sigma = (A \times B, X, f)$ be an asynchronous machine with n states and an adversarial input. Let w be an adversarial input character of Σ , and let $R(m, \Sigma, w)$ be the matrix of m stable transitions of Σ . Then, the following two statements are equivalent for all integers $m \geq n-1$ and all $i, j = 1, 2, \dots, n$.

- (i) The entry $R_{ij}(m, \Sigma, w)$ includes a string $w|u$ with $u \neq N$.
- (ii) The entry $R_{ij}^*(\Sigma, w)$ includes a string $w|u$ with $u \neq N$. $\blacklozenge$

In brief terms, Lemma 4.11 indicates that all stable transitions of the machine Σ with the adversarial input character w are characterized by the matrix $R(m, \Sigma, w)$, as long as $m \geq n-1$.

Of course, the adversarial input character is usually not known. To accommodate this fact, we introduce the following notion.

(4.12) DEFINITION. Let Σ be an asynchronous machine with the adversarial input uncertainty v . The *one-step stable transitions matrix* $\rho(\Sigma, v)$ of Σ consists of the entries

$$\rho_{ij}(\Sigma, v) := v_{w \in v} \rho_{ij}(\Sigma, w), \quad i, j = 1, 2, \dots, n. \quad \blacklozenge$$

The matrix $\rho(\Sigma, v)$ simply includes all one-step stable transitions that are compatible the adversarial uncertainty of Σ .

4.3. State Feedback

Recall that our objective is to build a state feedback controller C that turns the closed loop machine Σ_c of diagram (1.1) into a deterministic machine not affected by the adversarial input. The following notion forms the basis of our forthcoming discussion (compare to VENKATRAMAN and HAMMER [2006b and c]).

(4.13) DEFINITION. Let $\Sigma = (A \times B, X, f, v)$ be an asynchronous machine with an adversarial input, and let $x^i, x^j \in X$ be two states of Σ . We say that there is a *feedback path* from x^i to x^j if there is a state feedback controller that takes Σ from a stable combination with x^i to a stable combination with x^j in fundamental mode, given only that the adversarial input is within the uncertainty set v . $\blacklozenge$

In these terms, our objective is to find pairs of states of the machine Σ that can be connected by a feedback

path. Note that, due to fundamental mode operation, the adversarial input character does not change along a feedback path, but its value is not, in general, known. The following is a simple property of feedback paths.

(4.14) LEMMA. Let $\Sigma = (A \times B, X, f)$ be an asynchronous machine with the matrix of stable transitions $R(m, \Sigma, v)$. Let $x^i, x^j \in X$ be two states of Σ , and assume that there is a feedback path from x^i to x^j . If w is a possible adversarial input value along this path, then $w \in \Pi_a R_{ij}(m, \Sigma, v)$ for some $m \geq 1$.

Proof. Assume that there is a feedback path from x^i to x^j for the adversarial input value w . Then, there is a control input string that takes Σ from x^i to x^j through a string of stable transitions, while the adversarial input character is w . In view of Lemma 4.11, this implies that there is a control input string $u \in A^+$ such that $w|u \in R_{ij}(m, \Sigma, v)$ for some $m \geq 1$. Consequently, $w \in \Pi_a R_{ij}(m, \Sigma, v)$, and our proof concludes. ♦

Lemma (4.14) provides a simple necessary condition for the existence of a feedback path. We derive a sufficient condition for the existence of feedback paths in the next section.

5. COMPLETE SETS OF STRINGS

Let $\Sigma = (A \times B, X, f, v)$ be an asynchronous machine with an adversarial input, and let x^i and x^j be two states of Σ . In this section, we develop a test to determine whether there is a feedback path from x^i to x^j . Critical to this development is the uncertainty about the adversarial input value. This uncertainty may vary along a feedback path due to the fact that the controller accumulates more information about the adversarial input value.

As an example, assume that v consists of two characters, say $v = \{w^1, w^2\}$. Then, initially, it is known only that the adversarial input value is one of the characters w^1 or w^2 . Now, assume that the control input value is changed to the character u' . Letting s be the stable recursion function of Σ , we have two options for the next stable state:

- (i) $x' := s(x, u', w^1)$ when the adversarial input character is w^1 ; and
- (ii) $x'' := s(x, u', w^2)$ when the adversarial input character is w^2 .

Clearly, if $x' \neq x''$, then we can determine the value of the adversarial input character from the next stable state, thus resolving the uncertainty. On the other hand, if $x' = x''$, then the outcome of this step does not reduce the uncertainty about the adversarial input.

In summary, the uncertainty about the adversarial input value may be reduced as we progress along a feedback path. Of course, only the uncertainty changes - the adversarial input value itself is constant along a feedback path in fundamental mode operation.

The adversarial uncertainty affects the selection of the next control input character, as we now discuss. Consider the case where the machine Σ is at a stable combination with the state x and the control input value u_0 , while being driven along a feedback path toward a stable combination with the state x' . Let $v_0 \subset v$ be the current uncertainty about the value of the adversarial input, and let A be the control input alphabet. Let S be the set of all strings that take Σ from its current state x to a stable combination with the state x' , i.e., all strings $w|u = w|u_0 u_1 \dots \in v_0 | A^+$ for which $s(x, u, w) = x'$. Letting u_1 be the next control input character, denote by $S(a)$ the subset of all strings of S for which $u_1 = a$, where $a \in A$ is a character of the control input alphabet. Then, the set of adversarial input characters that are compatible with the control input character a at step 1 is given by $\Pi_a S(a)$.

Now, if $v_0 \not\subset \Pi_a S(a)$, then the character a cannot be used as the control input at step 1, since it is not compatible with some adversarial input values that may presently be active. On the other hand, if $v_0 \subset \Pi_a S(a)$, then a can be applied as the next control input character, since it is compatible with the information currently available about the adversarial input value. To conclude, the current level of adversarial uncertainty v_0 impacts the selection of the next control input character.

For a member $\sigma = w|u_0 u_1 \dots u_k$ of S and an integer $q \geq 0$, it is convenient to define the truncated string

$$\sigma|_q := \begin{cases} w|u_0u_1\dots u_q & \text{if } q \leq k, \\ w|u_0u_1\dots u_k & \text{if } q > k. \end{cases}$$

The set of all truncated members of S is denoted by

$$S|_q := \{\sigma|_q : \sigma \in S\}, q = 1, 2, \dots$$

Recall that S includes all strings that take Σ from a stable combination with the state x to a stable combination with the state x' . Then, the string $\sigma|_q$ takes Σ to a stable combination with the state

$$x_q := s(x, \sigma|_q) := s(x, u_0u_1\dots u_q, w), q = 1, 2, \dots$$

The stable states that Σ passes while being driven by the string σ are given by the list

$$x_0(\sigma) := s(x, \sigma|_0), x_1(\sigma) := s(x, \sigma|_1), \dots, x_k(\sigma) := s(x, \sigma|_k),$$

where $x_0(\sigma) = x$ and $x_k(\sigma) = x'$.

For a string $\sigma = w|u_0u_1\dots u_k \in S$, we define the projection $\Pi^p : S \rightarrow A$ which extracts the p -th control input character of σ , i.e.,

$$\Pi^p \sigma := \begin{cases} u_p & \text{for } p = 0, 1, \dots, k, \\ u_k & \text{for all } p > k. \end{cases}$$

Next, assume that the machine Σ is operated by a state feedback controller C that uses strings from the set S to drive Σ , while the adversarial input value w is kept constant. As usual, there is no direct information about the value of w . However, the control input values of Σ and the states through which Σ passes are known to the controller, as the controller generates the input values and reads the states of Σ . This data can be used to reduce the uncertainty about the adversarial input value w , as follows.

Let x and x' be two states of Σ , let u be a control input string of Σ , and let s be the stable recursion function of Σ . Assume that Σ is in a stable combination with the state x when the control input value changes to u , and let x'' be the next stable state of Σ . Define the *adversarial inverse function* s^a by setting

$$(5.1) \quad s^a(x, u, x'') := \{w \in B : s(x, u, w) = x''\},$$

so that $s^a(x, u, x'')$ is the set of all adversarial input values $w \in v$ that are compatible with the stable transition $s(x, u, w) = x''$. In particular, when Σ is at a stable combination with the initial state x_0 and the control input value u_0 , it follows from (5.1) that the adversarial input character w must satisfy

$$(5.2) \quad w \in v(x_0, u_0) := s^a(x_0, u_0, x_0) \cap v.$$

Thus, the initial uncertainty about the adversarial input value may, in fact, be smaller than v .

Recall that S is the set of all strings that take Σ from a stable combination with the state $x_0 := x$ to a stable combination with the state x' . At the initial step, the set of all possible adversarial input values is given by (5.2). Consequently, the set S must contain a path for each adversarial input character $w \in v(x_0, u_0)$, namely, we must have $v(x_0, u_0) \subset \Pi_a S$. Otherwise, the set S would be incompatible with some of the potential adversarial input values.

Further, let u_1 be a control input character, and let $S(x_0, u_0u_1)$ be the set of all strings of S whose control input starts with u_0u_1 , i.e.,

$$S(x_0, u_0u_1) = \{\sigma \in S : \sigma|_1 = w|u_0u_1 \text{ for some } w \in B\}.$$

Clearly, the character u_1 can be used as the next control input only if it is compatible with all possible adversarial input values, i.e., only if

$$v(x_0, u_0) \subset \Pi_a S(x_0, u_0u_1).$$

As the control input string is generated by the controller C , the pair (x_0, u_1) must be detectable to facilitate fundamental mode operation of the closed loop machine.

Now, let x_1 be the next stable state of Σ reached with the control input character u_1 . The fact that Σ has

reached the state x_1 implies that the adversarial input value w must have been within the set

$$v(x_0x_1, u_0u_1) := s^a(x_0, u_1, x_1) \cap v(x_0, u_0).$$

Continuing in this way, suppose that we are at step p of the path. Let $u_0u_1\dots u_p$ be the control input characters applied so far to Σ along this path by the controller, and let $x_0x_1\dots x_p$ be the string of stable states through which Σ has passed as a result. Let $v(x_0x_1\dots x_p, u_0u_1\dots u_p) \subset B$ be the current uncertainty about the adversarial input value. By iterating the earlier step, we obtain the following conclusion.

(5.3) LEMMA. Let w be the adversarial input character of the asynchronous machine Σ and let $p \geq 1$ be an integer. Assume that the control input string $u_0u_1\dots u_p \in A^+$ drives Σ through the states $x_0x_1\dots x_p$, where (x_i, u_i, w) , $i = 0, 1, 2, \dots, p$, are all stable combinations. Then, $w \in v(x_0x_1\dots x_p, u_0u_1\dots u_p)$, where

$$v(x_0x_1\dots x_p, u_0u_1\dots u_p) := s^a(x_{p-1}, u_p, x_p) \cap v(x_0x_1\dots x_{p-1}, u_0u_1\dots u_{p-1}). \blacklozenge$$

Referring to Lemma (5.3), it follows from (5.2) that $v(x_0x_1\dots x_p, u_0u_1\dots u_p) \subset v$. We call $v(x_0x_1\dots x_p, u_0u_1\dots u_p)$ the *residual adversarial uncertainty* at step p .

Let us return now to our set of strings S that take the machine Σ from the state $x_0 := x$ to the state x' . Denote by $S(x_0x_1\dots x_p, u_0u_1\dots u_p)$ the set of all elements $\sigma \in S$ that satisfy the following conditions:

- (i) The control input values are $u_0u_1\dots u_p$; and
- (ii) The machine Σ passes through the states $x_0, x_1, \dots, x_p$.

For a control input character $d \in A$, denote by $S(x_0x_1\dots x_p, u_0u_1\dots u_p d)$ the set of all strings $\sigma \in S(x_0x_1\dots x_p, u_0u_1\dots u_p)$ that have the character d in position $p+1$ of their control input string. Applying Lemma 5.3 to step p of the machine Σ , it follows that the adversarial input value must be within the set $v(x_0x_1\dots x_p, u_0u_1\dots u_p)$. Also, the set of all adversarial input characters that appear with the next control input character d is $\Pi_a S(x_0x_1\dots x_p, u_0u_1\dots u_p d)$. Combining the last two facts, we obtain the following.

(5.4) LEMMA. The character $d \in A$ can be used as the next control input character of the machine Σ only if $v(x_0x_1\dots x_p, u_0u_1\dots u_p) \subset \Pi_a S(x_0x_1\dots x_p, u_0u_1\dots u_p d)$. $\blacklozenge$

The condition of Lemma 5.4 is critical to the construction of a feedback controller that automatically takes a machine Σ with an adversarial input from one specified state to another. In fact, we show later that this condition guaranties the existence of such a controller, if it is valid at every stable transition along the way from x_0 to x' . These considerations lead us to the following.

(5.5) DEFINITION. Let $S \subset B|A^+$ be a set of strings taking the asynchronous machine Σ from a stable combination with the state x_0 to a stable combination with the state x' . The set S is *complete* if the following two conditions hold for all integers $p = 0, 1, 2, \dots$ and for every control input character $d \in \Pi^{p+1} S(x_0x_1\dots x_p, u_0u_1\dots u_p)$:

- (i) $v(x_0x_1\dots x_p, u_0u_1\dots u_p) \subset \Pi_a S(x_0x_1\dots x_p, u_0u_1\dots u_p d)$, and
- (ii) The pair (x_p, d) is detectable with respect to the residual adversarial uncertainty $v(x_0x_1\dots x_p, u_0u_1\dots u_p)$. $\blacklozenge$

Our next objective is to show that the existence of a complete set of strings is equivalent to the existence of a state feedback controller. Shortly thereafter, we present an algorithm for the derivation of complete sets of strings. First, we show that a complete set of strings can be replaced by a complete set of strings of bounded length. For a set of strings $S \subset B|A^+$, denote by $|S|$ the maximal length of a control input string in S , i.e., the maximal length of a string of the set $\Pi_c S$. For a finite set Z , denote by $\#Z$ the number of elements of Z .

(5.6) LEMMA. Let $\Sigma = (A \times B, X, f, v)$ be an asynchronous machine with n states, and let x_0 and x' be two states of Σ . Assume that Σ is in a stable combination at the state x_0 with the control input value u_0 . If there is a complete set of strings from x_0 to x' , then there also is such a complete set of strings S satisfying $|S| \leq [\#v(x_0, u_0)](n-1)$.

Proof. Consider a string $\sigma \in S$. Let $u = u_0u_1\dots u_k = \Pi_c \sigma$ be the control input values of this string and let $x_0x_1\dots x_k$ be the string of stable states through which Σ passes as a result of receiving the control input string u . The residual adversarial uncertainty at the start of the path is $v_0 := v(x_0, u_0)$. Let v_i be the residual uncertainty at step i of the path, and note that, by definition, v_i is a monotone declining function of i , and its minimal value cannot be less than 1. Divide the interval $[0, k]$ into segments of constant residual uncertainty. This results in the set of $m+1$

subintervals $I = \{[0, i_1], [i_1+1, i_2], \dots, [i_m+1, k]\}$, where v_i is constant over each one of these intervals. Since v_i is a monotone declining function and its minimum cannot be less than 1, we get $m+1 \leq \#v(x_0, u_0)$, or $m \leq \#v(x_0, u_0) - 1$.

Now, if any of the subintervals $[i, i'] \in I$ has length $\ell \geq n$, then the string of states $x_i x_{i+1} \dots x_{i'}$ must contain a repeating state, say $x := x_p = x_r$, where $i \leq p < r \leq i + \ell$. Since $v_p = v_r$ by construction, the control input value u_p can be replaced by the control value u_r without disturbing the stable combination at step p (recall that the adversarial input value is constant during the entire path). Then, steps $p+1, p+2, \dots, r$ can be eliminated from the string, resulting in a new segment with the length of $\ell - (r - p)$. This process can be repeated again and again, until the length of the resulting segment is less than n . Applying the same procedure to each one of the segments in I , we obtain a new path of length not exceeding $(m+1)(n-1) \leq [\#v(x_0, u_0)](n-1)$. As this bound is valid for every segment in I , our proof concludes. ♦

We have reached the main result of this section.

(5.7) THEOREM. Let $\Sigma = (A \times B, X, f, v)$ be an asynchronous machine and let x and x' be two states of Σ . Then, the following two statements are equivalent.

- (i) There is a state feedback controller C that drives Σ from a stable combination with x to a stable combination with x' in fundamental mode operation.
- (ii) There is a complete set of strings $S \subset B|A^+$ taking Σ from a stable combination with x to a stable combination with x' .

Proof. Assume first that (ii) is valid. We build a state feedback controller $F(x, x', v)$ which, upon receiving the input character $v \in A$, generates a string of control input characters that takes Σ from a stable combination with $x_0 := x$ to a stable combination with x' in fundamental mode operation. To this end, assume that Σ is in a stable combination with the state x_0 , and pick a control input character $u_1 \in \Pi^1 S$. Due to the fact that S is a complete set of strings, the pair (x_0, u_1) is detectable with respect to the adversarial uncertainty $v(x_0, u_0)$. In addition, $v(x_0, u_0) \subset \Pi_a S(x_0, u_0, u_1)$, so that the input character u_1 is compatible with every possible adversarial input value.

Now, let Ξ be the state set of the controller $F(x, x', v)$. The recursion function ϕ of $F(x, x', v)$ has three variables: the state of $F(x, x', v)$, the state of Σ , and the external control input, i.e., $\phi : \Xi \times X \times A \rightarrow \Xi$. Denote by $\eta : \Xi \times X \times A \rightarrow A$ the output function of $F(x, x', v)$, and let ξ_0 be the initial state of $F(x, x', v)$. We construct next the functions ϕ and η .

Upon encountering a detectable transition of Σ to the state x_0 with the control input value u_0 , the controller moves to a stable combination with the state ξ_1 . This transition prepares the controller to generate the input string that will take Σ to the state x' , when commanded to do so; it is accomplished by setting

$$\begin{aligned} \phi(\xi_0, (z, t)) &:= \xi_0 \text{ for all } (z, t) \neq (x_0, u_0), \\ \phi(\xi_0, (x_0, u_0)) &:= \xi_1, \\ \phi(\xi_1, (x_0, u_0)) &:= \xi_1. \end{aligned}$$

While in its initial state ξ_0 or in the state ξ_1 , the controller applies to the control input of Σ the external input character it receives, namely

$$\begin{aligned} \eta(\xi_0, (z, t)) &:= t \text{ for all } (z, t) \in X \times A, \\ \eta(\xi_1, (z, t)) &:= t \text{ for all } (z, t) \in X \times A, \end{aligned}$$

so that $F(x, x', v)$ is transparent in these states.

Suppose now that, while $F(x, x', v)$ is in the state ξ_1 , it receives the external input character $v \in A$. This is the command for the controller to start a string of transitions taking Σ from its current stable combination with the state x_0 to a stable combination with the state x' . Upon receiving the external input value v , the controller $F(x, x', v)$ moves to a stable combination with the state ξ_2 , namely,

$$\begin{aligned} \phi(\xi_1, (z, t)) &:= \xi_1 \text{ for all } (z, t) \neq (x_0, v), \\ \phi(\xi_1, (x_0, v)) &:= \xi_2, \\ \phi(\xi_2, (x_0, v)) &:= \xi_2. \end{aligned}$$

When reaching the state ξ_2 , the controller applies to the control input of Σ the first character of the control input string $u_1 u_2 \dots u_k \in \Pi_c S$ that ultimately takes Σ to the state x' ; to this end, set

$$\eta(\xi_2, (x_0, t)) := u_1 \text{ for all } t \in A.$$

The control input character u_1 causes Σ to move to the state x_1 through a detectable transition, since S was a complete set of strings. For the same reason, u_1 is compatible with every adversarial input character in the residual adversarial uncertainty $v(x_0, u_0)$. As the transition was detectable, Σ is in a stable combination when it reaches the state x_1 . When the controller detects the state x_1 of Σ , it moves to a stable combination with the state ξ_3 , namely,

$$\phi(\xi_2, (z, t)) := \xi_2 \text{ for all } (z, t) \neq (x_1, u_1),$$

$$\phi(\xi_2, (x_1, v)) := \xi_3,$$

$$\phi(\xi_3, (x_1, v)) := \xi_3.$$

When reaching the state ξ_3 , the controller applies to Σ the next control input value u_2 . Since S is a complete set of strings, the pair (x_1, u_2) is detectable for the current adversarial uncertainty $v(x_0 x_1, u_0 u_1)$. Also, $v(x_0 x_1, u_0 u_1) \subset \Pi_a S(x_0 x_1, u_0 u_1)$, so that u_2 is compatible with every possible adversarial input value. We then build the controller output function accordingly:

$$\eta(\xi_3, (x_1, t)) := u_2 \text{ for all } t \in A.$$

Continuing in this manner, assume that the controller $F(x, x', v)$ has so far generated the control input string $u_0 u_1 \dots u_p$, taking Σ through the states $x_0 x_1 \dots x_p$; here, p is an integer between 1 and k . Since S was a complete set of strings, the transition to the state x_p was a detectable transition; consequently, Σ is in a stable combination when it reaches the state x_p . Upon detecting the state x_p , the controller $F(x, x', v)$ moves to a stable combination with the state ξ_{p+2} , namely,

$$\phi(\xi_{p+1}, (z, t)) := \xi_{p+1} \text{ for all } (z, t) \neq (x_p, u_p),$$

$$\phi(\xi_{p+1}, (x_p, v)) := \xi_{p+2},$$

$$\phi(\xi_{p+2}, (x_p, v)) := \xi_{p+2}.$$

Now, select any control input value $u_{p+1} \in \Pi^{p+1} S(x_0 x_1 \dots x_p, u_0 u_1 \dots u_p)$. Due to the fact that S is a complete set of strings, the pair (x_p, u_{p+1}) is detectable with respect to the adversarial uncertainty $v(x_0 x_1 \dots x_p, u_0 u_1 \dots u_p)$. Also, $v(x_0 x_1 \dots x_p, u_0 u_1 \dots u_p) \subset \Pi_a S(x_0 x_1 \dots x_p, u_0 u_1 \dots u_p u_{p+1})$, so that the input character u_{p+1} is compatible with any adversarial character in $v(x_0 x_1 \dots x_p, u_0 u_1 \dots u_p)$. Upon reaching the state ξ_{p+2} , the controller applies to Σ the control input character u_{p+1} , namely,

$$\eta(\xi_{p+2}, (x_p, t)) := u_{p+1} \text{ for all } t \in A.$$

This construction is repeated for $p = 1, 2, \dots$, until the machine Σ reaches the state x' . In view of our construction, the resulting controller $F(x, x', v)$ satisfies condition (i) of the Theorem. Note that, by Lemma 5.6, the state x' can always be reached at a step $k \leq (n - 1)[\#v(x_0, u_0)]$, where n is the number of states of the machine Σ .

Conversely, assume that condition (i) is valid. Let $F(x, x', v)$ be a controller which, upon receiving the input character $v \in A$, takes Σ from a stable combination with the state $x_0 := x$ to a stable combination with the state x' . Assume that Σ is in a stable combination with the state x_0 and the control input value u_0 , when the controller input changes to the character v . The initial uncertainty about the adversarial input character of Σ is then $v(x_0, u_0)$. Let $S \subset B|A^+$ be the set of strings the controller $F(x, x', v)$ can generate; the specific string used by $F(x, x', v)$ in each case depends on the information it extracts about the adversarial input value. To prove that (i) implies (ii), we need to show that S is a complete set of strings. Recall that, due to fundamental mode operation, the adversarial input character w , although possibly unknown, remains constant until Σ reaches the stable state x' .

To show that S is a complete set of strings, consider step $p \geq 0$ of a string of control input characters $u_0 u_1 \dots u_p$ applied by $F(x, x', v)$ to Σ . Denote by $x_0 x_1 \dots x_p$ the stable states through which Σ has passed as a result of this string; here, x_p is the current stable state of Σ . By Lemma 5.3, the residual adversarial uncertainty at this point is $v(x_0 x_1 \dots x_p, u_0 u_1 \dots u_p)$. Let $d \in A$ be the next control input character that F generates for Σ . Then, by fundamental mode operation of the closed loop machine, the pair (x_p, d) is detectable with respect to the residual uncertainty $v(x_0 x_1 \dots x_p, u_0 u_1 \dots u_p)$. Also, in view of Lemma 5.4, we have $v(x_0 x_1 \dots x_p, u_0 u_1 \dots u_p) \subset \Pi_a S(x_0 x_1 \dots x_p, u_0 u_1 \dots u_p d)$. As this

is true for all $p \geq 0$, the requirements of Definition 5.5 are met, and S is a complete set of strings. Thus, (i) implies (ii), and our proof concludes. ♦

In view of Theorem 5.7, finding a complete set of strings is the critical step in the process of designing a controller for an asynchronous machine with adversarial input. The following algorithm derives such a set of strings.

(5.8) ALGORITHM. Derivation of a complete set of strings. Let $\Sigma = (A \times B, X, f, v)$ be an asynchronous machine with adversarial input. Let $S \subset B|A^+$ be a set of strings that take Σ from a stable combination with the state x_0 and the control input value u_0 to a stable combination with the state x' . Consider the case where Σ is at step j of an input string from S , having received the control inputs $u_0 u_1 \dots u_j$ and having passed through the stable states $x_0 x_1 \dots x_j$. The residual adversarial uncertainty is $v(x_0 x_1 \dots x_j, u_0 u_1 \dots u_j)$.

Step 0. Set $j := 0$.

Step 1. If $v(x_0 x_1 \dots x_j, u_0 u_1 \dots u_j) \not\subset \Pi_a S(x_0 x_1 \dots x_j, u_0 u_1 \dots u_j)$, then S does not include a complete set of strings. Set $\Phi := \emptyset$ and terminate the algorithm. Otherwise, continue to Step 2.

Step 2. Let S_1 be the set of all strings $\sigma \in S(x_0 x_1 \dots x_j, u_0 u_1 \dots u_j)$ for which $\Pi_a \sigma \not\subseteq v(x_0 x_1 \dots x_j, u_0 u_1 \dots u_j)$. If $S_1 \neq \emptyset$, then replace S by the difference set $S \setminus S_1$ and go to Step 0. If $S_1 = \emptyset$, continue to Step 3.

Step 3: Let S_2 be the set of all strings $\sigma \in S(x_0 x_1 \dots x_j, u_0 u_1 \dots u_j)$ for which the pair $(x_j, \Pi^{j+1} \sigma)$ is not detectable with respect to the residual uncertainty $v(x_0 x_1 \dots x_j, u_0 u_1 \dots u_j)$. If $S_2 \neq \emptyset$, then replace S by the difference set $S \setminus S_2$ and go to Step 0. If $S_2 = \emptyset$, continue to Step 4.

Step 4. Let S_3 be the set of all strings $\sigma \in S(x_0 x_1 \dots x_j, u_0 u_1 \dots u_j)$ for which $v(x_0 x_1 \dots x_j, u_0 u_1 \dots u_j) \not\subset \Pi_a S(x_0 x_1 \dots x_j, u_0 u_1 \dots u_j \Pi^{j+1} \sigma)$. If $S_3 \neq \emptyset$, then replace S by the difference set $S \setminus S_3$ and go to Step 0. If $S_3 = \emptyset$, continue to Step 5.

Step 5. Let q be the length of the longest string in S . If $j = q$, then set $\Phi := S$ and terminate the algorithm. Otherwise, replace j by $j+1$ and go to Step 1. ♦

The outcome of the Algorithm 5.8 is a set of strings $\Phi \subset B|A^+$. If Φ is not the empty set, then, according to the next statement, it forms a complete set of strings.

(5.9) THEOREM. Let $\Sigma = (A \times B, X, f, v)$ be an asynchronous machine with adversarial input, let $S \subset B|A^+$ be a set of strings all having the same initial control input character, and let Φ be the outcome of Algorithm 5.8. Then,

(i) Φ is not empty if and only if S contains a complete set of strings, and

(ii) If Φ is not empty, then it forms a complete set of strings included in S .

Proof. Assume that Φ is not empty. Then, an examination of Step 3 of Algorithm 5.8 shows that Φ satisfies condition (i) of Definition 5.5, while an examination of Step 4 of the Algorithm shows that Φ satisfies condition (ii) of Definition 5.5. A slight reflection on the flow of Algorithm 5.8 leads then to the conclusion that Φ is a subset of S , and that it is not empty if and only if S contains a complete set of strings. ♦

6. SKELETON MATRICES

We turn now to the definition of one of the main notions of our present discussion. Let $\Sigma = (A \times B, X, f, v)$ be an input/state asynchronous sequential machine with adversarial input, having the state set $X = \{x^1, \dots, x^n\}$ with n states. In view of (5.2), the initial adversarial uncertainty always satisfies $v(x_0, u_0) \subset v$, so we always have $\#v(x_0, u_0) \leq \#v$. Invoking Lemma 5.6, we conclude that a complete set of strings S for the machine Σ can always be selected so that its length satisfies

$$(6.1) \quad |S| \leq (n-1)(\#v).$$

Recall that, by (4.10), the i, j entry of the matrix $R(m, \Sigma, w)$ includes all the control input strings that take Σ from a stable combination with the state x^i to a stable combination with the state x^j through a string of m or fewer stable transitions, while the adversarial input character is w . At this point, it is convenient to introduce the

matrix

$$(6.2) \quad R(\Sigma, \nu) := \nu_{w \in \nu} R((n-1)(\# \nu), \Sigma, w).$$

(6.3) DEFINITION. The matrix $R(\Sigma, \nu)$ is the *combined matrix of stable transitions* of the asynchronous machine Σ with the adversarial uncertainty ν . ♦

(6.4) EXAMPLE. Continuing with our analysis of the machine Σ of Example 2.2, we let the adversarial uncertainty be $\nu = \{\alpha, \beta\}$, so that $\# \nu = 2$. As we have three states in this case, $n = 3$, and $(n-1)(\# \nu) = 4$.

$$R(\Sigma, \nu) = \rho(\Sigma, \alpha) \vee \rho(\Sigma, \beta) \vee \rho^2(\Sigma, \alpha) \vee \rho^2(\Sigma, \beta) \vee \rho^3(\Sigma, \alpha) \vee \rho^3(\Sigma, \beta) \vee \rho^4(\Sigma, \alpha) \vee \rho^4(\Sigma, \beta) =$$

$$\left(\begin{array}{ccc} \{\alpha|a, \alpha|aba, \alpha|baba, \alpha|ba, \alpha|abab, \alpha|ab, \alpha|bab, \alpha|b, \beta|a, \beta|baba, \beta|aba, \beta|ba\} & \{\beta|abab, \beta|bab, \beta|ab, \beta|b\} & \{\alpha|N, \beta|N\} \\ \{\alpha|baba, \alpha|aba, \alpha|abab, \alpha|bab, \alpha|ab, \beta|a, \beta|baba, \beta|aba, \beta|ba\} & \{\alpha|b, \beta|abab, \beta|bab, \beta|ab, \beta|b\} & \{\alpha|ba, \alpha|a\} \\ \{\alpha|baba, \alpha|ba, \alpha|aba, \alpha|abab, \alpha|bab, \alpha|b, \alpha|ab, \beta|baba, \beta|ba, \beta|aba\} & \{\beta|abab, \beta|bab, \beta|b, \beta|ab, \beta|a\} & \{\alpha|a, \beta|a\} \end{array} \right) \diamond$$

Considering (6.1), Lemma 5.6, and Theorem 5.7, we reach the following conclusion.

(6.5) COROLLARY. Let $\Sigma = (A \times B, X, f, \nu)$ be an asynchronous machine with adversarial input, having the state set $X = \{x^1, \dots, x^n\}$ and the combined matrix of stable transitions $R(\Sigma, \nu)$. Then, the following two statements are equivalent for all $i, j = 1, 2, \dots, n$.

- (i) There is a state feedback controller that takes Σ from a stable combination with x^i to a stable combination with x^j in fundamental mode operation.
- (ii) The i, j entry of $R(\Sigma, \nu)$ includes a complete set of strings. ♦

In view of Corollary 6.5, it is easy to determine whether or not there is a state feedback controller that takes the machine Σ from a stable combination with the state x^i to a stable combination with the state x^j in fundamental mode operation: all we have to do is apply Algorithm 5.8 to the entry $R_{ij}(\Sigma, \nu)$. Then, such a controller exists if and only if the outcome of Algorithm 5.8 is a non empty set. This set can then be used to construct an appropriate controller by following the proof of Theorem 5.7. In this way, we arrive at the following notion.

(6.6) DEFINITION. Let $\Sigma = (A \times B, X, f, \nu)$ be an asynchronous machine with adversarial input, having n states and the combined matrix of stable transitions $R(\Sigma, \nu)$. The *complete matrix of stable transitions* $\mathfrak{R}(\Sigma, \nu)$ of Σ is an $n \times n$ matrix defined as follows for each $i, j \in \{1, 2, \dots, n\}$: the entry $\mathfrak{R}_{ij}(\Sigma, \nu)$ is a complete set of strings included in the entry $R_{ij}(\Sigma, \nu)$; if there is no such complete set, then $\mathfrak{R}_{ij}(\Sigma, \nu) := N$. ♦

(6.7) EXAMPLE. Applying Algorithm 5.8 on the entries of the matrix of stable transitions derived in Example 6.4, we obtain

$$\mathfrak{R}(\Sigma, \nu) = \left(\begin{array}{ccc} \{\alpha|a, \beta|a\} & N & N \\ \{\alpha|ab, \beta|a, \beta|ba\} & \{\alpha|b, \beta|ab, \beta|b\} & N \\ \{\alpha|ab, \alpha|ba, \alpha|b, \beta|ba\} & N & \{\alpha|a, \beta|a\} \end{array} \right) \diamond$$

In view of Theorem 5.7, the following is true.

(6.8) COROLLARY. Let Σ be an asynchronous sequential machine with the state set $\{x^1, \dots, x^n\}$ and the adversarial uncertainty ν . Let $\mathfrak{R}(\Sigma, \nu)$ be the complete matrix of stable transitions of Σ . Then, the following two statements are equivalent for all $i, j \in \{1, \dots, n\}$:

- (i) There is a state feedback controller that takes Σ from a stable combination with the state x^i to a stable combination with the state x^j in fundamental mode operation.
- (ii) $\mathfrak{R}_{ij}(\Sigma, \nu) \neq N$. ♦

We can now generalize the notion of the skeleton matrix (MURPHY, GENG, and HAMMER [2002 and 2003]) to asynchronous machines with adversarial inputs.

(6.9) DEFINITION. Let $\Sigma = (A \times B, X, f, \nu)$ be an input/state asynchronous sequential machine with adversarial input, having the state set $X = \{x^1, \dots, x^n\}$ and the complete matrix of stable transitions $\mathfrak{R}(\Sigma, \nu)$. The *control skeleton*

matrix $K(\Sigma, \nu)$ of Σ is an $n \times n$ matrix of zeros and ones, whose entries are defined as follows for each $i, j \in \{1, 2, \dots, n\}$: $K_{ij}(\Sigma, \nu) := 1$ if $\mathfrak{R}_{ij}(\Sigma, \nu) \neq N$, and $K_{ij} := 0$ if $\mathfrak{R}_{ij}(\Sigma, \nu) = N$. ♦

(6.10) EXAMPLE. Using the result of Example 6.7, we obtain the control skeleton matrix

$$K(\Sigma, \nu) = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \text{ ♦}$$

In these terms, a state feedback controller can take Σ from a stable combination with the state x^i to a stable combination with the state x^j in fundamental mode operation if and only if $K_{ij}(\Sigma, \nu) = 1$.

6.1. Latent Adversarial Switches.

We turn our attention now to a restricted version of the model matching problem with adversarial inputs. The solution of this problem forms a step stone along our way toward the solution of the full model matching problem with adversarial inputs.

Consider an asynchronous machine Σ with adversarial input uncertainty ν . Note that changes in the adversarial input value do not always cause a state transition of Σ . Indeed, assume that Σ is at a stable combination with the state x and the control input character u_0 , and consider the set of adversarial input characters $s^a(x, u_0, x)$ of (5.1). Clearly, if this set consists of more than one character, then a switch of the adversarial input from one character to another in this set does not change the stable state of Σ , and hence is not noticeable by a state feedback controller. This leads to the following.

(6.11) DEFINITION. A *latent adversarial switch* is a change of the adversarial input value that does not result in a change of the stable state of the machine. ♦

The next statement provides a solution of the model matching problem for the case when all adversarial input changes are latent.

(6.12) THEOREM. Let $\Sigma = (A \times B, X, f, \nu)$ be an input/state machine with adversarial input, and assume that the adversarial input is restricted to latent switches. Let $K(\Sigma, \nu)$ be the control skeleton matrix of Σ , and let $\Sigma' = (A, X, s')$ be a stable-state input/state machine with no adversarial input, having the skeleton matrix $K(\Sigma')$. Then, the following two statements are equivalent.

- (i) There exists a state feedback controller C for which $\Sigma_{c|s} = \Sigma'$, where the closed loop machine Σ_c is well posed and operates in fundamental mode.
- (ii) $K(\Sigma, \nu) \geq K(\Sigma')$.

(6.13) EXAMPLE. Consider the problem of building a model matching controller for the machine Σ of Example 2.2 so as to match the model Σ' of Example 3.2. Using the procedure described in MURPHY, GENG, and HAMMER [2002 and 2003], the skeleton matrix of the model Σ' is calculated as

$$K(\Sigma') = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

The control skeleton matrix $K(\Sigma, \nu)$ was calculated in Example 6.10. Comparing the two matrices, we obtain that $K(\Sigma, \nu) \geq K(\Sigma')$, so that a model matching controller exists by Theorem 6.12. We will construct the controller in section 9 below. ♦

Proof (of Theorem 6.12). Assume first that (i) is valid, and consider a particular stable transition of the model Σ' , say a transition from a stable combination with the state x^i to a stable combination with the state x^j . Then, by definition of the skeleton matrix, $K_{ij}(\Sigma') = 1$. As (i) is valid, the stable state machine $\Sigma_{c|s}$ must also have a transition from a stable combination with the state x^i to a stable combination with x^j . In view of the fact that Σ_c is the machine Σ controlled by the controller C , it follows by Corollary 6.8 and Definition 6.9 that $K_{ij}(\Sigma, \nu) = 1$ as well. Thus, $K_{ij}(\Sigma, \nu) = 1$ if $K_{ij}(\Sigma') = 1$, $i, j = 1, 2, \dots, n$. Considering that $K(\Sigma, \nu)$ and $K(\Sigma')$ are both matrices of zeros

and ones, the latter implies that $K(\Sigma, v) \geq K(\Sigma')$, and (i) implies (ii).

Conversely, assume that (ii) is valid. Let $K^1(\Sigma')$ be the one-step skeleton matrix of the model Σ' (see MURPHY, GENG, and HAMMER [2003]). Then, $K(\Sigma') \geq K^1(\Sigma')$, and it follows by (ii) that

$$(6.14) \quad K(\Sigma, v) \geq K^1(\Sigma').$$

Now, let $i, j \in \{1, 2, \dots, n\}$ be a pair of integers for which $K_{ij}^1(\Sigma') = 1$. Let s' be the stable recursion function of the model Σ' , and denote by $V(i, j)$ the set of all control input characters $v \in A$ for which $s'(x^i, v) = x^j$. Note that, for all i ,

$$(6.15) \quad V(i, j) \cap V(i, j') = \emptyset \text{ if } j \neq j',$$

since different target states require different inputs. As $K_{ij}^1(\Sigma') = 1$, it follows by (6.14) that also $K_{ij}(\Sigma, v) = 1$. By Corollary 6.8, there is then a complete set of strings from the state x^i to the state x^j . Select a character $v \in V(i, j)$. In view of Theorem 5.7, there is controller $F(x^i, x^j, v)$ that takes the machine Σ from a stable combination with x^i to a stable combination with x^j ; here, the character v activates the controller. Extend the activation of this controller to all characters $v \in V(i, j)$, so that any character $v \in V(i, j)$ can be used to start the (same) controller action. Denote the resulting controller by $F(x^i, x^j, V(i, j))$.

Next, we define the following operation of *join* for combining two controllers (see also VENKATRAMAN and HAMMER [2006c]). Given two controllers $F(x^i, x^j, V(i, j))$ and $F(x^{i'}, x^{j'}, V(i', j'))$, the *join*

$$C := F(x^i, x^j, V(i, j)) \vee F(x^{i'}, x^{j'}, V(i', j'))$$

is constructed as follows:

(i) When $x^i = x^{i'}$ and the external input character is v , then

$$C := \begin{cases} F(x^i, x^j, V(i, j)) & \text{if } v \in V(i, j), \\ F(x^{i'}, x^{j'}, V(i', j')) & \text{if } v \in V(i', j'); \end{cases}$$

(ii) If $(x^i, x^j) \neq (x^{i'}, x^{j'})$, let the machine Σ be in a stable combination with the state x^i , when the controller C receives the input character v . Then,

$$C := F(x^i, x^j, V(i, j)).$$

This construction of C is consistent by (6.15) (see VENKATRAMAN and HAMMER [2006c] for more details).

Now, let $T \subset \{1, 2, \dots, n\} \times \{1, 2, \dots, n\}$ be the set of all pairs of integers for which $K_{ij}^1(\Sigma') = 1$. Then, a slight reflection shows that the joined controller

$$F := \vee_{i,j \in T} F(x^i, x^j, V(i, j))$$

makes the machine Σ match the model Σ' . Theorem 5.7, the closed loop machine is well defined and operates in fundamental mode. This concludes our proof. ♦

7. GENERAL MODEL MATCHING

7.1. Adversarial Detectability

Let $\Sigma = (A \times B, X, f, v)$ be an asynchronous machine with adversarial input. Assume that Σ is at a stable combination (x, u, w) , when the adversarial input value changes to w' . This change may or may not cause Σ to experience a transition to a new stable state. Presently, consider the case when the change in the adversarial input from w to w' causes Σ to move to a new state $x' \neq x$. We refer to such a transition as an *adversarial transition*. In this section, we discuss the existence and the design of a state feedback controller C^a that automatically counteracts adversarial transitions of Σ .

A basic requirements is, of course, that the controller C^a operate in fundamental mode to guaranty

deterministic behavior of the closed loop machine. Being a state feedback controller, C^a has access to the current state of Σ , and it generates the control input of Σ . To obtain fundamental mode operation of the closed loop machine with the controller C^a , it must be possible to determine from the state of Σ whether or not Σ has reached its next stable combination. This leads us to the following, which is closely analogous to Definition 4.4.

(7.1) DEFINITION. Let $\Sigma = (A \times B, X, f)$ be an input/state asynchronous machine with adversarial uncertainty ε . Assume that Σ is in a stable combination at the state x with the control input character u , when a change in the adversarial input causes Σ to move to the state x' . Then, the pair (x, u) is *adversarially detectable* with respect to the adversarial uncertainty ε if it can be determined from the current state of Σ whether or not Σ has reached its next stable combination. ♦

Without adversarial detectability, it is not possible to guaranty fundamental mode operation of a closed loop machine controlled by a state feedback controller. In other words, operation must be restricted to adversarially detectable pairs of the controlled machine Σ .

Assume then that the machine Σ is at a stable combination (x, u, w) , when the adversarial input character changes to w' , causing Σ to transition to a stable combination with the state $x' \neq x$. This transition may, of course, consist of a number of intermediate steps, say $x_0 := x$, $x_1 := f(x_0, u, w')$, $x_2 = f(x_1, u, w')$, ..., $x_q := f(x_{q-1}, u, w') = x'$, $x_q := f(x_q, u, w')$. Similarly to (4.1) and (4.2), we denote

$$(7.2) \quad \begin{cases} \theta(x, u, w') := x_1 \dots x_q, \\ \theta[x, u, \varepsilon] := \{\theta(x, u, w') : w' \in \varepsilon\}. \end{cases}$$

The following statement is closely analogous to Theorem 4.5 and has a similar proof.

(7.3) THEOREM. Let $\Sigma = (A \times B, X, f, v)$ be an input/state asynchronous machine with adversarial input. Assume that Σ is in a stable combination with the state x and the control input value u . In the notation of (3.3) and (7.2), the following two statements are equivalent.

- (i) The pair (x, u) is adversarially detectable with respect to the adversarial uncertainty v .
- (ii) States of the set $s^v(x, u)$ appear only at the end of strings belonging to $\theta[x, u, v]$. ♦

As indicated earlier, to guaranty fundamental mode operation of the closed loop machine, the use of the machine Σ must be restricted to adversarially detectable pairs. This leads us to the following notion. (For a string $\sigma = w|u_1u_2\dots u_q \in B \times A^+$, denote by $\Pi_c^+ \sigma := u_q$ the last control input character of the string.)

(7.4) DEFINITION. Let $\Sigma = (A \times B, X, f, v)$ be an asynchronous machine having adversarial uncertainty v , n states, and the combined matrix of stable transitions $R(\Sigma, v)$. The *reduced matrix of stable transitions* $R^l(\Sigma, v)$ of Σ is obtained by removing from each column $j = 1, 2, \dots, n$ of $R(\Sigma, v)$ all strings σ for which the pair $(x^j, \Pi_c^+ \sigma)$ is not adversarially detectable with respect to the uncertainty v . ♦

(7.5) EXAMPLE. We calculate now the reduced matrix of stable transitions for the matrix $R(\Sigma, v)$ of Example 6.4. Considering the transition table of the machine Σ as provided in Example 2.2, note that there is only one transition that can be caused by the adversarial input, namely, the transition initiated by a switch of the adversarial input from the character α to the character β , while Σ is at the state x^1 and the control input is b . Symbolically, the transition can be represented by $(x^1, b, \alpha) \rightarrow (x^1, b, \beta) \rightarrow (x^2, b, \beta)$. For this transition, recalling that s is the stable recursion function of Σ , we have $s(x^1, b, \beta) = x^2$. Hence, $\theta[x^1, b, \alpha] = x^1$ and $\theta[x^1, b, \beta] = x^2$, so that

$$\theta[x^1, b, v] = \{x^1, x^2\}.$$

In this case, we have

$$s^v(x^1, b) = \{x^1, x^2\}.$$

As we can see, the states x^1, x^2 appear only at the end of strings belonging to $\theta[x^1, b, v]$. Therefore, by Theorem 7.3, the pair (x^1, b) is adversarially detectable, and we have $R^l(\Sigma, v) = R(\Sigma, v)$ in this case. ♦

The reduced matrix of stable transitions characterizes all transitions of the machine Σ that end at adversarially detectable pairs. This matrix forms the basis for designing controllers that can counteract adversarial transitions.

7.2. Reversing Adversarial Transitions.

Assume that the machine Σ is at an adversarially detectable stable combination with the pair $(x^s, u) \in X \times A$, when a change in the adversarial input causes Σ to move to a stable combination with the state x^t ; of course, by fundamental mode operation, the control input value u is kept constant during this process. As this transition started from an adversarially detectable pair, a state feedback controller can determine from the current state of Σ whether or not Σ has reached its next stable combination. Having been caused by the adversarial input, this transition is undesirable; our objective is to design a state feedback controller C^a that automatically reverses this transition. In this way, C^a will counteract the effects of the adversarial input.

Consider then an adversarial transition from a stable combination with the pair (x^s, u) to a stable combination with the pair (x^t, u) . Letting v be the adversarial uncertainty, the set of adversarial input characters that can give rise to such a transition is

$$(7.6) \quad v(x^s, x^t, u) := s^a(x^s, u, x^t) \cap v.$$

Clearly, this transition is possible if and only if $v(x^s, x^t, u) \neq \emptyset$, and we reach the following.

(7.7) DEFINITION. Let Σ be an asynchronous machine with the state set $X = \{x^1, x^2, \dots, x^n\}$ and the adversarial uncertainty v , and assume that Σ is in a stable combination with the control input character u . Then, for a pair of integers $s, t \in \{1, 2, \dots, n\}$, the *adversarial transition indicator* is

$$(7.8) \quad K(x^s, x^t, u) := \begin{cases} 1 & \text{if } v(x^s, x^t, u) \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases} \quad \blacklozenge$$

The discussion leading to Definition 7.7 implies the next statement.

(7.9) LEMMA. Assume that the closed loop machine Σ is in a stable combination at the state x^s with the control input character u . Then, the following two statements are equivalent.

- (i) There is an adversarial transition to a stable combination with the state x^t .
- (ii) $K(x^s, x^t, u) = 1$. $\blacklozenge$

To address the question of whether an adversarial transition is reversible or not, it is convenient to introduce the following.

(7.10) DEFINITION. Let Σ be an asynchronous machine with adversarial input. An adversarial transition from a stable combination with the state x^s to a stable combination with the state x^t is *reversible* if there is a state feedback controller that drives Σ back to a stable combination with the state x^s , without specific information about the adversarial character that caused the transition. $\blacklozenge$

To examine reversible transitions, assume that the machine Σ is in a stable combination with the state x^s and the control input character u , when an adversarial transition to the state x^t occurs. In view of (7.6), the adversarial uncertainty immediately after the transition is $v(x^s, x^t, u)$. Using the reduced matrix of stable transitions $R^r(\Sigma, v)$, we construct the scalar function

$$(7.11) \quad K^r(x^s, x^t, u) := \begin{cases} 1 & \text{if } R_{ts}^r(\Sigma, v) \text{ includes a complete set of strings} \\ & \text{with respect to the adversarial uncertainty } v(x^s, x^t, u), \\ 0 & \text{otherwise.} \end{cases}$$

A slight reflection indicates that the following is true.

(7.12) LEMMA. In the above notation, an adversarial transition from x^s to x^t is reversible if and only if $K^r(x^s, x^t, u) = 1$. $\blacklozenge$

Next, let $U(x^s) \subset A$ be the set of all control input characters of the machine Σ that may appear in stable combinations with the state x^s . For various practical considerations, the designer of the state feedback controller C may choose to avoid using some of these control input characters. The *set of active control input characters* of Σ at the state x^s is the subset $S(x^s) \subset U(x^s)$ of all control input characters that may be applied by the controller C , while the closed loop machine Σ_c is in a stable combination at the state x^s of Σ . If x^s does not appear as part of a

stable state of the closed loop machine, then $S(x^s) := \emptyset$. When all states of Σ appear as stable states of the closed loop machine Σ_c and all possible control input characters of Σ are utilized, we have

$$(7.13) \quad S(x^s) = U(x^s), s = 1, 2, \dots, n.$$

Equality (7.13) describes the most common situation. When it is valid, we say that there are *no idle control input characters*.

(7.14) DEFINITION. Let Σ be an asynchronous machine with adversarial input having the n states $x^1, \dots, x^n$. Let $S(x^s)$ be the set of active control input characters at the state x^s . The *reversal matrix* $A(S, \Sigma)$ is an $n \times n$ numerical matrix with the entries

$$A_{st}(S, \Sigma) := \begin{cases} \min \{K^r(x^s, x^t, u) - K(x^s, x^t, u) : u \in S(x^s)\} & \text{if } S(x^s) \neq \emptyset, \\ 0 & \text{if } S(x^s) = \emptyset, \end{cases}$$

$s, t = 1, 2, \dots, n$. ♦

For a numerical matrix D , the inequality $D \geq 0$ means that D has no negative entries.

(7.15) EXAMPLE. We continue our examination of the machine Σ of Example 2.2. In view of Example 7.5, there is only one adversarial transition we have to consider, namely, the transition $x^2 = s(x^1, b, \beta)$. Assume that the controller is implemented with no idle control input characters, so that $S(x^1) = U(x^1)$. From (7.6), we have

$$v(x^1, x^2, b) = s^a(x^1, b, x^2) \cap v = \{\beta\} \cap \{\alpha, \beta\} = \{\beta\}.$$

From (7.8), we get that $K(x^1, x^2, b) = 1$. In Example 7.5, we have seen that $R^r(\Sigma, v) = R(\Sigma, v)$. Also, from Example 6.7, we have that the complete set of strings $\mathfrak{R}_{21}(\Sigma, v) = \{\alpha|ab, \beta|a, \beta|ba\}$. Consequently, $R_{21}^r(\Sigma, v)$ includes a complete set of strings with respect to the adversarial uncertainty $v(x^1, x^2, b) = \{\beta\}$. Substituting into (7.11), we obtain that $K^r(x^1, x^2, b) = 1$, so that $A_{12}(U, \Sigma) = 0$. As there are no adversarial transitions other than the transition from x^1 to x^2 , we have that $K(x^s, x^t, u) = 0$ for all $(s, t) \neq (1, 2)$. Whence, $A(U, \Sigma) \geq 0$ in this case. ♦

We can characterize now the conditions under which a transition caused by the adversarial input can be counteracted by a state feedback controller.

(7.16) THEOREM. Let Σ be an asynchronous machine with adversarial input, and let $X = \{x^1, \dots, x^n\}$ be the state set of Σ . Assume that Σ is operated by a state feedback controller using the active control input character sets $S(x^s)$, $s = 1, 2, \dots, n$, and let $A(S, \Sigma)$ be the corresponding reversal matrix. Then, the following two statements are equivalent.

- (i) All adversarial transitions of the closed loop machine can be automatically reversed in fundamental mode operation.
- (ii) $A(S, \Sigma) \geq 0$.

Proof. Consider an adversarial transition from a stable combination with the state x^s to a stable combination with the state x^t . We have two cases here, depending on the set $S(x^s)$ of active control input characters:

Case 1: $S(x^s) = \emptyset$: then, the state x^s does not appear in a stable combination of the closed loop machine, and hence no adversarial transitions can start at the state x^s . Then, $A_{st}(S, \Sigma) = 0$ by Definition 7.14.

Case 2: $S(x^s) \neq \emptyset$: By Lemma 7.9, an adversarial transition from x^s to x^t is possible if and only if $K(x^s, x^t, u) = 1$ for a control input value $u \in S(x^s)$. By Lemma 7.12, this transition can be reversed if and only if $K^r(x^s, x^t, u) = 1$. As $K(x^s, x^t, u)$ and $K^r(x^s, x^t, u)$ can only take the values 0 or 1, we conclude that the adversarial transition from x^s to x^t is reversible if and only if $K^r(x^s, x^t, u) - K(x^s, x^t, u) \geq 0$. As this is true for all states x^s and x^t and for all input values $u \in S(x^s)$, it follows that (i) and (ii) are equivalent. This concludes our proof. ♦

Most often, a state feedback controller employs every control input character of the set $U(x^s)$. In such case, the combination of Theorems (6.12) and (7.16) yields the following statement, which is the main result of this section.

(7.17) THEOREM. Let $\Sigma = (A \times B, X, f, v)$ be an input/state machine with adversarial input, and assume that Σ has no idle control input characters. Let $U(x)$ be the set of control input characters that appear in stable combinations with the state x , let $A(U, \Sigma)$ be the corresponding reversal matrix, and let $K(\Sigma, v)$ be the control skeleton matrix of Σ . Let $\Sigma' = (A, X, s')$ be a stable-state input/state machine with no adversarial input, having the skeleton matrix

$K(\Sigma')$. Then, the following two statements are equivalent:

- (i) There is a controller C for which $\Sigma_{c|s} = \Sigma'$, where Σ_c is well posed and operates in fundamental mode.
- (ii) $K(\Sigma, v) \geq K(\Sigma')$ and $A(U, \Sigma) \geq 0$. ♦

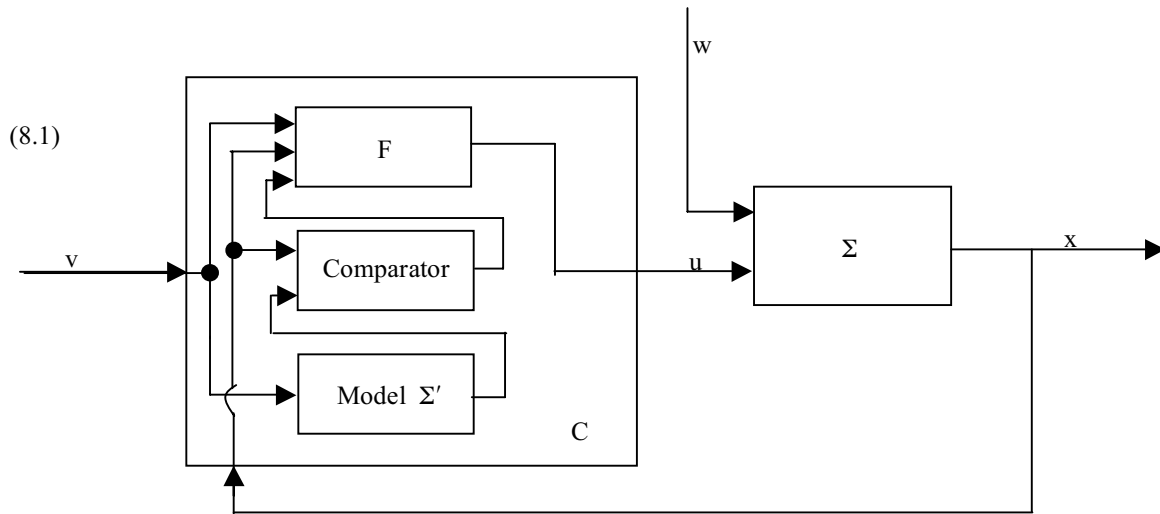
Theorem 7.17 provides a comprehensive solution to the model matching problem for asynchronous machines with adversarial inputs. As we can see, the solution is entirely characterized by two numerical matrix inequalities. As the model machine Σ' has no adversarial input, the controller C , when it exists, counteracts any effects of adversarial activity.

(7.18) EXAMPLE. Combining now the results of Examples 6.13 and 7.15, we conclude that condition (ii) of Theorem 7.17 is valid for the machine Σ of Example 2.2 and the model Σ' of Example 3.2. Therefore, Theorem 7.17 assures us that there is a controller C that controls Σ so that the closed loop machine Σ_c is stably equivalent to the model Σ' , thus solving the perturbed model matching problem in this case. The construction of the controller C is described in section 9 below. ♦

8. CONTROLLER STRUCTURE

We summarize now the structure of a controller that solves the model matching problem with adversarial inputs. In general terms, the controller consists of two components: a component that achieves model matching and a component that reverses adversarial transitions.

Consider the problem of controlling the machine Σ to match the model Σ' . The construction of the model matching component of the controller is described in the proof of Theorem 6.12. Regarding adversarial transitions - these, by their nature, occur while the model Σ' remains in a stable combination, since the model has no adversarial input. Therefore, an adversarial transition can be characterized as follows (refer to the diagram below): it is a departure from a stable combination of the closed loop Σ_c that is not preceded by a change of the external command input v . We describe next the controller in brief terms.



- (i) The state feedback controller F controls the machine Σ to achieve model matching in response to the external command input v . The controller F is built following the procedure described in the proof of Theorem 6.12.
- (ii) The comparator detects stable states of Σ_c that differ from stable states of Σ' . When such a difference is detected, the comparator activates the state feedback controller F to drive Σ back to a stable combination with the correct state.

An example of the construction of the controller C is provided in the next section.

9. EXAMPLE

In this section, we construct a controller C that solves the model matching problem for the machine Σ of Example 2.2 and the model Σ' of Example 3.2; the adversarial uncertainty is $v = \{\alpha, \beta\}$. It can be shown that condition (ii) of Theorem 7.17 is satisfied in this case. Below, we demonstrate the construction of a model matching controller. Recall that s is the stable transition function of Σ and s' is the stable transition function of Σ' . An examination of the transition tables of the machines Σ and Σ' shows that only the following three transitions of Σ are different from corresponding transitions of Σ' :

$$\begin{aligned} s(x^1, b, \beta) &= x^2 \quad :: \quad s'(x^1, b) = x^1; \\ s(x^2, a, \alpha) &= x^3 \quad :: \quad s'(x^2, a) = x^1; \\ s(x^3, b, \beta) &= x^2 \quad :: \quad s'(x^3, b) = x^1. \end{aligned}$$

Using the procedure described in the proof of Theorem 6.12, we build three state feedback controllers $F(x^1, x^1, b)$, $F(x^2, x^1, a)$, and $F(x^3, x^1, b)$, each of which respectively "corrects" one of these transitions.

1) Construction of the controller $F(x^1, x^1, b)$:

We use the state set $\{\xi^0, \xi^1, \xi^2, \xi^3\}$ for $F(x^1, x^1, b)$. Note that (x^1, a) is a detectable pair. Upon detecting a detectable transition of Σ to the state x^1 with the control input value a , the controller $F(x^1, x^1, b)$ moves to a stable combination with its state ξ^1 , while continuing to apply to Σ the input character it receives:

$$\begin{aligned} \phi(\xi^0, (z, t)) &:= \xi^0 \text{ for all } (z, t) \neq (x^1, a), \\ \phi(\xi^0, (x^1, a)) &:= \xi^1, \\ \phi(\xi^1, (x^1, a)) &:= \xi^1, \\ \eta(\xi^0, (z, t)) &:= t \text{ for all } (z, t) \in X \times A, \\ \eta(\xi^1, (z, t)) &:= t \text{ for all } (z, t) \in X \times A, \end{aligned}$$

Next, upon receiving the external input character b , the controller $F(x^1, x^1, b)$ moves to a stable combination with its state ξ^2 , namely:

$$\begin{aligned} \phi(\xi^1, (z, t)) &:= \xi^1 \text{ for all } (z, t) \neq (x^1, b), \\ \phi(\xi^1, (x^1, b)) &:= \xi^2, \\ \phi(\xi^2, (x^1, b)) &:= \xi^2. \end{aligned}$$

At this point, the controller $F(x^1, x^1, b)$ must start to generate a string of control input characters to keep Σ at the state x^1 . An examination of the entry $\mathfrak{N}_{11}(\Sigma, v)$ in Example 6.7 shows that the single control input character a satisfies this requirement. So we set the output function of the controller $F(x^1, x^1, b)$ as

$$\eta(\xi^2, (x^1, t)) := a \text{ for all } t \in A.$$

When the controller $F(x^1, x^1, b)$ detects the state x^1 of Σ , it moves to a stable combination with the state ξ^3 , namely,

$$\begin{aligned} \phi(\xi^2, (z, t)) &:= \xi^2 \text{ for all } (z, t) \neq (x^1, a), \\ \phi(\xi^2, (x^1, b)) &:= \xi^3, \\ \phi(\xi^3, (x^1, b)) &:= \xi^3, \end{aligned}$$

and continues to generate the control input character a :

$$\eta(\xi^3, (x^1, t)) := a \text{ for all } t \in A.$$

Finally, upon a change of the external input character, $F(x^1, x^1, b)$ resets to its initial state ξ^0 :

$$\phi(\xi^3, (z, t)) := \xi^0 \text{ for all } (z, t) \neq (x^1, b).$$

2) The construction of the controllers $F(x^2, x^1, a)$ and $F(x^3, x^1, b)$ is similar to the construction of $F(x^1, x^1, b)$.

3) Counteracting adversarial transitions:

Recall from Example 7.15 that the machine Σ has only one transition that can be caused by the adversarial input - the transition $x^2 = s(x^1, b, \beta)$ when the adversarial input character changes from α to β . Assume then that

the machine Σ is at a stable combination with the pair (x^1, b) , when the state of Σ switches to x^2 . By Example 7.15, the adversarial uncertainty is then $v(x^1, x^2, b) = \{\beta\}$. We build now a controller $F^a(x^2, x^1)$ that counteracts this action of the adversarial input, and returns Σ to the state x^1 immediately after the comparator detects the transition to x^2 . The output d of the comparator can take two values: $d := 1$ when the stable state of Σ_c is different from the stable state of Σ' , and $d := 0$ when the two stable states are equal.

Referring to Examples 6.7, 7.5, and 7.15, we have that the complete set of strings $\{\alpha|ab, \beta|a, \beta|ba\}$ is included in the entry $\mathcal{R}_{21}(\Sigma, v)$; that $R^r(\Sigma, v) = R(\Sigma, v)$; and that $v(x^1, x^2, b) = \{\beta\}$. Consequently, the single character a forms a control input string that takes Σ back to the state x^1 . The construction of the controller $F^a(x^2, x^1)$ is reminiscent of the construction of the controller $F(x^1, x^1, b)$ we have described earlier, and is as follows. Let φ be the recursion function of $F^a(x^2, x^1)$, let μ be its output function, and let $\{\zeta^0, \zeta^1, \zeta^2\}$ be its state set. Starting at a stable combination with the pair (x^2, b) , set

$$\begin{aligned} \varphi(\zeta^0, (z, t, 0)) &:= \zeta^0 \text{ for all } (z, t) \neq (x^2, b); \\ \mu(\zeta^0, (z, t)) &:= t \text{ for all } (z, t) \in X \times A; \\ \varphi(\zeta^0, (x^2, b, 1)) &:= \zeta^1; \\ \varphi(\zeta^1, (x^2, b, d)) &:= \zeta^1, d \in \{0, 1\}; \\ \mu(\zeta^1, (z, t)) &:= a \text{ for all } (z, t) \in X \times A; \\ \varphi(\zeta^1, (x^1, b, d)) &:= \zeta^2, d \in \{0, 1\}; \\ \varphi(\zeta^2, (x^1, b, d)) &:= \zeta^2, d \in \{0, 1\}; \\ \mu(\zeta^2, (z, t)) &:= a \text{ for all } (z, t) \in X \times A; \\ \varphi(\zeta^2, (z, t, d)) &:= \zeta^0 \text{ for all } (z, t) \neq (x^1, b). \end{aligned}$$

Finally, we assemble the combined state feedback controller F by using the join operation employed in the proof of Theorem 6.12:

$$F = F(x^1, x^1, b) \vee F(x^2, x^1, a) \vee F(x^3, x^1, b) \vee F^a(x^2, x^1).$$

When this controller is inserted into the control diagram (8.1), it eliminates the effects of the adversarial input and makes Σ behave like the deterministic and unperturbed model Σ' of Example 3.2.

10. REFERENCES

G. BARRETT and S. LAFORTUNE

[1998] "Bisimulation, the supervisory control problem, and strong model matching for finite state machines," Discrete Event Dynamic Systems: Theory and Application, vol. 8, no. 4, pp. 377–429.

M. D. DIBENEDETTO, A. SALDANHA and A. SANGIOVANNI-VINCENTELLI

[1994] "Model matching for finite state machines," Proceedings of the IEEE Conference on Decision and Control, vol. 3, 1994, pp. 3117–3124.

S. EILENBERG

[1974] "Automata, Languages and Machines," Academic Press, New York.

X. J. GENG and J. HAMMER

[2004] "Asynchronous sequential machines: input/output control," Proceedings of the 12th Mediterranean Conference on Control and Automation, Kusadasi, Turkey, June 2004.

[2005] "Input/output control of asynchronous sequential machines," IEEE Transactions on Automatic Control, vol. 50, no. 12, pp. 1956–1970.

J. HAMMER

[1994] "On some control problems in molecular biology," Proceedings of the IEEE conference on Proceedings and Control, December 1994.

[1995] "On the modeling and control of biological signal chains," Proceedings of the IEEE conference on Decision and Control, December 1995.

[1996a] "On the corrective control of sequential machines," International Journal of Control, vol. 65, no. 2, pp. 249–

276.

[1996b] "On the control of incompletely described sequential machines," International Journal of Control, vol. 63, no. 6, pp. 1005-1028.

[1997] "On the control of sequential machines with disturbances," International Journal of Control, vol. 67, no. 3, pp. 307-331.

Z. KOHAVI

[1970] "Switching and Finite Automata Theory," McGraw-Hill Book Company, New York.

T. E. MURPHY, X. J. GENG and J. HAMMER

[2002] "Controlling races in asynchronous sequential machines," Proceeding of the IFAC World Congress, Barcelona, July 2002.

[2003] "On the control of asynchronous machines with races," IEEE Transactions on Automatic Control, vol. 48, no. 6, pp. 1073-1081.

P. J. G. RAMADGE and W. M. WONHAM

[1987] "Supervisory control of a class of discrete event processes," SIAM Journal of Control and Optimization, vol. 25, no. 1, pp. 206-230.

J. G. THISTLE and W. M. WONHAM

[1994] "Control of infinite behavior of finite automata," SIAM Journal on Control and Optimization, vol. 32, no. 4, pp. 1075-1097.

N. VENKATRAMAN and J. HAMMER

[2006a] "Stable realizations of asynchronous sequential machines with infinite cycles," Proceedings of the 2006 Asian Control Conference, Bali, Indonesia, 2006.

[2006b] "Controllers for asynchronous machines with infinite cycles," Proceedings of the 17th International Symposium on Mathematical Theory of Networks and Systems, Kyoto, Japan, 2006.

[2006c] "On the control of asynchronous sequential machines with infinite cycles," International Journal of Control, vol. 79, no. 7, pp. 764-785.