LOCAL BEHAVIOR IDENTIFICATION OF GENERATIVE NETWORK AUTOMATA

A MASTER'S THESIS

in

Computer Engineering

Atılım University

by

BURAK ÖZDEMİR

JULY 2010

GCRIS

LOCAL BEHAVIOR IDENTIFICATION OF GENERATIVE NETWORK AUTOMATA

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

ATILIM UNIVERSITY

BY

BURAK ÖZDEMİR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF COMPUTER ENGINEERING

JULY 2010

GCRIS

Approval of the Graduate School of Natural and Applied Sciences, Atılım University.

_____

Prof. Dr. İbrahim AKMAN

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof. Dr. İbrahim AKMAN

Head of Department

This is to certify that we have read the thesis "Local Behavior Identification of Generative Network Automata" submitted by "Burak ÖZDEMİR" and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Asst. Prof. Dr. Hürevren KILIÇ

Supervisor

Examining Committee Members

Asst. Prof. Dr. Hakan TORA _____

Asst. Prof. Dr. Mehmet CANTÜRK _____

Asst. Prof. Dr. Gökhan ŞENGÜL _____

Asst. Prof. Dr. Erol ÖZÇELİK _____

Asst. Prof. Dr. Hürevren KILIÇ _____

Date: 01.07.2010

GCRIS

I declare and guarantee that all data, knowledge and information in this document has been obtained, processed and presented in accordance with academic rules and ethical conduct. Based on these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

BURAK ÖZDEMİR

GCRIS

ABSTRACT

LOCAL BEHAVIOR IDENTIFICATION OF GENERATIVE NETWORK AUTOMATA

Özdemir, Burak

M.S., Computer Engineering Department

Supervisor: Assist.Prof.Dr. Hürevren Kılıç

July 2010, 45 pages

Relation between the whole and its parts is investigated in the context of complex discrete dynamical systems. For that purpose, an algorithm for local behavior identification from global data described as Generative Network Automata (GNA) model configurations is developed. It is shown that one can devise a procedure to simulate finite GNA configurations via Automata Networks having static rule-space setting. An abstraction which is called "overlay level component interaction" is introduced. Besides, the existence of a GNA identification procedure is shown. In practice, the algorithm provides an automated approach to model construction and it can suitably be used in GNA based system modeling effort.

Keywords: Generative Network Automata, Automata Networks, Inverse Problem, Design Identification, Discrete Dynamical Systems, Simulation, Artificial Life.

ÖZ

DOĞURGAN AĞ ÖZDEVİNİRLERİN YEREL DAVRANIŞ TANIMASI

Özdemir, Burak

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Yrd. Doç. Dr. Hürevren Kılıç

Temmuz 2010, 45 sayfa

Bir bütün ve onu oluşturan parçaların kendi aralarındaki ilişkisi, karmaşık ayrık dinamik sistemler bağlamında önem taşımaktadır. Bu amaçla, Doğurgan Ağ Özdeviniri şeklinde tanımlanan genel verilerden yerel davranış tanımlama algoritması geliştirilmiştir. Sabit kural-uzayı düzenlemesine sahip Özdevinir Ağları kullanarak sonlu Doğurgan Ağ Özdevinirlerinin tanınması ve benzetimi amacıyla bir izlek tasarlanabilirliği gösterilmiştir. "Üst seviye bileşen etkileşimi" adı verilen bir soyutlama sunulmuştur. Ayrıca, Doğurgan Ağ Özdevinirleri için bir tanımlama yönteminin varlığı gösterilmiştir. Pratikte, model inşasına yönelik otomatikleştirilmiş bir yaklaşım sağlanmıştır ve geliştirilen yaklaşım Doğurgan Ağ Özdevinirlerini esas alan sistem modelleme girişimlerinde kullanılabilmektedir.

Anahtar Kelimeler: Doğurgan Ağ Özdevinirleri, Özdevinir Ağları, Ters Problem, Tasarım Tanıma, Ayrık Dinamik Sistemler, Benzetim, Yapay Yaşam.

To My Parents

GCRIS

ACKNOWLEDGEMENTS

I am heartily thankful to my supervisor, Hürevren Kılıç, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. I also offer sincere thanks to my family, for their continuous support and patience. Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of this Thesis.

GCRIS

TABLE OF CONTENTS

ABSTRACT iii
ÖZ iv
DEDICATION v
ACKNOWLEDGMENTS vi
TABLE OF CONTENTS vii
LIST OF TABLES viii
LIST OF FIGURES ix
LIST OF ABBREVIATIONS xi

CHAPTER

1. INTRODUCTION 1
2. AUTOMATA NETWORKS 4
    2.1 Cellular Automata 5
    2.2 Non-Uniform Cellular Automata 9
    2.3 Random Boolean Networks 12
    2.4 Generative Network Automata 15
3. GENERATIVE NETWORK AUTOMATA IDENTIFICATION 19
    3.1 Definitions and Assumptions 19
    3.2 Algorithm 22
    3.2 Algorithm worst-case and best-case analysis 29
4. EXAMPLE APPLICATION: SIMPLE LEGO® DESIGN 32
5. CONCLUSIONS 40

REFERENCES 42

APPENDICES

A. PSEUDOCODE FOR ID_GNA ALGORITHM 44

LIST OF TABLES

TABLE

1. Description of Rule 56 7
2. An example for a lookup table 14
3. Some real life examples with concurrent state and topology changes due to network dynamics 16
4. List of possible [(Connectivity, State)] → [(Connectivity), (State)] based component behavior modes for minimal Memory restricted systems under study 25
5. User-guidance table for conflict resolution 33
6. From first step to fifth step for component 1's perspective 37
7. User-guidance table for Lego® bridge example 38
8. Identified rules for component 1's perspective 38

GCRIS

LIST OF FIGURES

FIGURE

1. 1D Automata Network 4
2. Two basic neighborhood descriptions 5
3. 2D Cellular Automata grid as a Torus-shape 7
4. 1D Circularly connected Cellular Automata neighborhood 7
5. Neighborhood description in Game of Life 8
6. Still-lives, first pattern is called block, second one is called beehive 9
7. Oscillators, first pattern is called beacon, second one is called toad 9
8. Langton's loop 11
9. Cancer Cells 12
10. Nonlocal neighborhood (Irregular) 13
11. A RBN topology 14
12. Extraction mechanism selects a sub-GNA and returns it 17
13. Evolving of the sub-GNA and replacement 17
14. Resultant GNA 17
15. An example system snapshot 22
16. Reader Mode 23
17. Writer Mode 23
18. Transfer Mode 24
19. A simple configuration sequence 26
20. Extraction Figure a and b 27
21. Extraction Figure c and d 28
22. Three Types Lego® Bricks 34
23. Lego® bridge design 35
24. Configuration of the system at time t=1 35
25. Configuration of the system at time t=2 36
26. Configuration of the system at time t=3 36
27. Configuration of the system at time t=4 36

28. Configuration of the system at time t=5..... 37
29. Overlay level neighborhood graph for Lego® Bridge Example..... 39

GCRIS

LIST OF ABBREVIATIONS

AN	-	Automata Networks
CA	-	Cellular Automata
GNA	-	Generative Network Automata
NCA	-	Non-uniform Cellular Automata
NHL	-	Non-Hodgkin Lymphoma
RBN	-	Random Boolean Networks

GCRIS

CHAPTER I

INTRODUCTION

Generative Network Automata (GNA) is a generalized variant of discrete dynamic system modeling tools [1]. Dynamic Systems are means of describing how one state transforms into another [2]. In Discrete Dynamic Systems, the transformation occurs in discrete time steps, it is also periodic, and that is the reason why it is called discrete [3]. GNA enables to build a model from scratch, which contains small pieces. These small pieces are capable of changing their individual connectivity and states [1]. The process of building models involves use of some predetermined rules that conduct the pieces how to change state and connectivity to put forth essential acumen and hereby whole model operates fittingly what it is designed for.

Models are built to study a particular subject. In general, building a model may not be straightforward. The significance of rules emerges at the time of modeling a specific entity. A rule in discrete dynamical systems and GNA context is a mapping from a state/connectivity pair to another state/connectivity pair [1]. Every piece in a model follows particular rules. From now on, “piece” will be called as “individual”. In our context, a model is a system that contains individuals, and these individuals behave according to rules. We need to have both individuals and rules to create a GNA model. Without a reliable automated approach, rule search is difficult. The process of identifying underlying rules of an already existing-model is called “Inverse Problem”, building models using specific rules is called “Forward Problem” [4,5,6]. Solving an inverse problem sheds light to underlying rules, and gives the instructions to build a model in forward direction. Inverse problem is finding the values for parameters of a model by the help of observed system data [7].

In some aspect, inverse problem solving is reverse-engineering of the forward approach [8]. If you consider a city growth model, which is initially begins on a small area with little population, we expect it to expand a larger area and the population will increase if the city is developing in time. If we are needed to understand and simulate a city growth with different initial configurations (land size/amount of population), we have to find the underlying rules of the city growth model, and if we have rules of such a model, then the rules may be applied to different initial configurations to obtain different simulation runs. For instance, rules of a city growth model may be applied to different city configurations for a comparative study. Such examples can be increased. Whenever inverse problem is solved, the forward execution becomes possible. Since we have the rules, we can build the model and simulate the system.

Automata Networks [9] is one of the root node (i.e. general approach) in the discrete dynamic system modeling framework tree [8]. Under this node, Cellular Automata (CA), Non-uniform Cellular Automata, Random Boolean Networks and other variants exist. However, Generative Network Automata introduces a one step further generalization over CA-based modeling frameworks. It allows us to describe changing connectivity of networks of automata. So structures with dynamic topologies become understandable and more complex models can be built on GNA framework. Most systems in real life have dynamic topology, consider the Internet. New nodes are added to the system in every minute and no one knows the exact topology, why? The answer is simple, because, it is huge and highly dynamic. How the topology of a network changes? Since, one or more nodes of the system change connectivity to other node or nodes. Clearly, connection topology can be described as automata neighborhood. The system supports that any individual in the system can interact freely by others. By this way social networks for example, can be modeled easily. Since GNA is a new mathematical tool in this area, practical applications have not been appeared so far. But it is prone to represent any dynamical system; from organisms to eco systems in the nature.

The scope of this thesis is to develop a GNA identification algorithm which solves inverse problem in Discrete Dynamical Systems context. A system can be described with a transfer function, a linear differential equation, an impulse response or a set of

state equations. Given an input to the system, determining the output of the system is identification [21]. However, beginning with an input and output sample, figuring out the description of the system is the inverse problem [22,23]. The output of the algorithm is identified GNA rules, which produces the desired output and so these rules can be considered as a system description. The algorithm is named "ID_GNA – Identify Generative Network Automata". This algorithm identifies these connections along with the state-changes of the discrete individuals that form the system. The algorithm provides an automated approach to rule-based model construction and it can suitably be used in GNA based system modeling. Identification procedure returns a GNA from sequence that describes global behavior of the Discrete Dynamical System under study. The algorithm takes a sequence of GNA configurations as an input parameter and returns a set of identified GNA rules. Configurations are the snapshot of the system in the discrete time steps and rules are the functions that relate changes of connectivity and state in discrete time steps. By these rules, we may apply them to different initial configurations to populate different simulation runs to observe the system in various scenarios (i.e. when rules are applied to different initial configurations, results are different) [10]. Note that for a given sequence data, there may be another rules other than the identified GNA rules which can produce given global sequence data. For example, one may use the rules of a contagion of bacteria group, to build a model to observe a particular bacteria specie growth on an environment, which may be pretty useful like a real bacteria-culture environment in labs. However, the true mechanism behind such growth may still be unidentified because of insufficient global behavior data. So identification of relations bears a high importance, and must be studied through modeling phase.

The thesis consists of five chapters to describe the core ideas behind the development of a GNA identification algorithm. In the second chapter, the modeling frameworks are explained in detail. In the third chapter the proposed identification algorithm is introduced. In the fourth chapter a possible application area (Lego® example) of the algorithm is discussed, and the last chapter concludes the study.

CHAPTER II

AUTOMATA NETWORKS

There are well-known variants of automata networks, each of which has a different characteristic. Beginning line for Automata Networks theory is as old as computers. It was first introduced to literature by John Von Neumann [10] and he can be thought as founding father of the underlying concepts. After the introduction of Automata Networks theory, computer scientists wanted to make use of general framework of complex systems to make any computation organized and consisted of primitive elements. By this way, they have wanted this organized structure to exhibit complex behavior and solve problems.

A typical Automata Network can be imagined as network of inter-connected nodes. The nodes of this network have neighborhood among themselves. The organization of these nodes can mimic a natural system or can be organized, artificially. That's why biological motivation is a starting point for most ideas. The huge application area of Automata Networks contains anything that can be encoded 0, 1 [9]. In an automata network, group of nodes change their states in discrete time steps. A node considers its neighbor(s)'s states before changing its state. Figure-1 shows a simple example for one dimensional automata network. An Automata Network consists of components which is called "automaton". An automaton has a transition rule which changes its state according to neighbor automaton state.

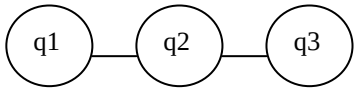


Figure1 – 1D Automata Network

State of automaton q1 affects automaton q2, automaton q2 affects automaton q3 and vice versa. In discrete time steps, each automaton situated at a node updates its state according to input feeds from its neighbors. Consider Figure-2 below for two different neighborhoods.

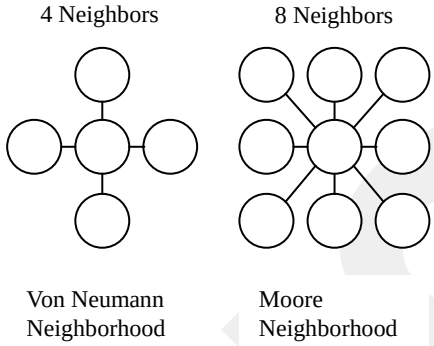


Figure2 –Two basic neighborhood descriptions.

The given neighborhood descriptions are fundamental ones, alternative even irregular neighborhood descriptions can also be generated. The neighborhood description has an indirect effect on the state space evolution of a network. It is related to complexity of the behavior that exhibited by the whole system.

Automata Network is a superset/root model for other variants like, Cellular Automata, Non-Uniform Cellular Automata, Hybrid Cellular Automata, Random Boolean Network [8] and even Generative Network Automata. Details of this general model are explained through its variants.

2.1 Cellular Automata

Cellular Automata is a discrete model, which is commonly used as a tool for discrete dynamical system simulations. It maintains a comprehensive environment via programmable cells. Cellular Automata was devised by John von Neumann [10]. He proposed Cellular Automata while working on self-replicating systems. His aim was to construct an entity that builds another one. So he needed a mathematical abstraction, and this was the born of Cellular Automata. Later he applied his design with environment rules and Universal Constructor emerged [10].

Cellular Automata is constituted from a grid of cells. Each cell has its own rule, which is a function and this function updates/decides the cell's next state. The other cells around a cell are neighbors of that cell. A cell has to care about the states of its neighbors and its own state, before updating its state, and the decision about how to update/decide next state is defined by the rule/function. Since Cellular Automata is a discrete model, it has to have an initial configuration at time t=0. By saying configuration, the states of the cells are intended. At time t=1, each cell will update its state by using its embedded rule (function). The input for this function is the states of the cells in the neighborhood (a cell is also supposed to be a neighbor of itself). The output is obviously the new state for the cell. A configuration at any time step is called a "generation", since a new configuration is formed from the previous one. Generally the rule is same for all the cells in the grid and it does not change. For the sake of clear understanding, let's define a simple rule, which may be as follows; when all the cells in the neighborhood are in "0" state, our cell will be in "1" state, and when all the cells in the neighborhood are in "1" state, our cell will be in "0" state. Note that the state transition rule can be partially defined.

Another concept in CA is dimension. If you consider a lattice like a square 2x2, 3x3, 4x4 or larger, it will be 2D model and this is more complex than 1D model. In 1D model, neighborhood is simpler, and it can be 1x2, 1x3, 1x4 or 1x***n*** (***n*** is a positive integer).

Consider a 3x3 lattice, and a 2-state cell (0 or 1) then, there are 2^9=512 different initial configurations. Each configuration maps to either 0 or 1. If you increase states, combinations will increase and mappings will differ (e.g. 3^9=19683 different configurations for a three-state CA). Neighborhood in Cellular Automata model is uniform, which means that any cell on the grid has the same number of neighbors and the same the neighborhood structure. This is uniform neighborhood concept, and it makes Cellular Automata a special instance of Automata Networks.

In general, Cellular Automata environment is a finite grid, and inherently we have a problem on the edge cells. The neighborhoods of the edge cells are handled in different ways. One can make the edge cells unchanging via fixed states, or can define special rules for these cells. There is another way in which we can morph our

2D grid as toroidal shape, join the top and bottom, then join left and right side of the grid and we get the shape in Figure-3.

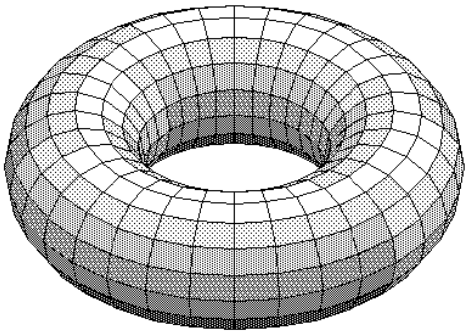


Figure 3 – 2D Cellular Automata grid as a Torus-shape.

So, we made the edge cells to be neighbors with themselves (cyclic). In 1D Cellular Automata, the solution is much simpler, the nearest left neighbor of the left most cell is the right most cell and the nearest right neighbor of the right most cell is the left most cell. In Figure-4, cell A and cell B are neighbors.

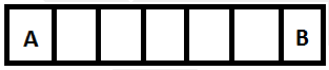


Figure 4 – 1D Circularly connected Cellular Automata neighborhood.

There are some common understandings/agreements over Cellular Automata model; the first one is Elementary Cellular Automata, which is 1 dimensional, 2 states, 3-neighborhood (including the cell itself). So, there are 2^3=8 different neighborhood patterns, and 2^8=256 different rules. The second convention is about rule enumeration. The value of the second row in Table-1 (i.e. binary 00111000 = decimal 56) is the enumerates rule number 56,

111	110	101	100	011	010	001	000
0	0	1	1	1	0	0	0

Table 1 – Description of Rule 56

According to Rule-56, whenever a cell on the grid has 101 as input pattern, its state value will be 1 on the next time step. Similarly, if it has 010 as input, the cell's next

state value becomes 0. Among 256 possible rules, rule 30 and rule 110 are interesting to analyze. They seem to exhibit random behaviors on the grid, meaning that their cycle lengths are high.

The most popular example for a cellular automaton is the Game of Life or Conway's Game of Life [13]. It was originated by the famous British Mathematician John Horton Conway. Let's define this model in our 2D Cellular Automata environment. In the Game of Life, each cell has 2 states, live or dead. Each cell interacts with its 8 neighbors which surrounds the cell.

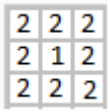


Figure 5 – Neighborhood description in Game of Life.

In Figure-5, the cells that are marked with 2 are the neighbors of cell which is marked with 1. The rules of this game are defined as follows:

I. Under population: A cell dies if it has less than 2 living neighbors.
II. Overcrowding: A cell dies if it has more than 3 living neighbors.
III. Next Generation: Any cell with 2 or 3 living neighbors lives in the next generation.
IV. Birth: Any dead cell with 3 live neighbors becomes a live cell.

Initially, there are some living cells in the grid (i.e. filled cells) and the above rules applied simultaneously to all cells in the grid, so next generation is created. Under some circumstances, patterns in the grid, stands still or loop. These two special cases are known as Still-lives and Oscillators. Still-lives patterns keep their shape through generations. Oscillators are similar to Still-lives patterns; the only difference is that they alternate between some patterns. Following Still-lives and Oscillator patterns are well-know examples for the Game of Life (see Figure-6 and Figure-7).

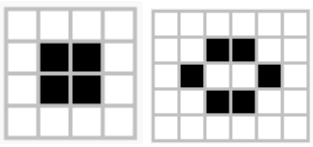


Figure 6 – Still-lives, first pattern is called *block*, second one is called *beehive*.

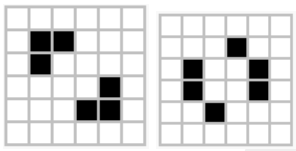


Figure 7 – Oscillators, first pattern is called *beacon*, second one is called *toad.*

In short, we can say that Cellular Automata environment enables us to define complex behaviors through simple rules.

2.2 Non-Uniform Cellular Automata

In previous section, we pointed out that states of cells are updated synchronously according to the rules of cells in Cellular Automata. But unlike ordinary Cellular Automata, the rules of cells can also change over time, i.e. there can be an evolution in rule space. As a result, cells will have time-varying rules, and this is where non-uniformity emerges. However, Non-Uniform Cellular Automata (NCA) keeps the basic properties of CA concept and there are still three important characteristics remain unchanged [14].

- *Parallelism*
- *Neighborhood Concept*
- *Simple Rules/Entities*

In addition to the above properties, we have changing rule space. Neighborhood configuration may be Von Neumann, Moore or irregular. If we consider a cell's rule as a genotypic characteristic, its behavioral consequences can be thought as

phenotypic characteristics. Furthermore a cell's rule changes according to some criteria's. These criteria's are explained below.

Dead or Live: A cell being in 1-state is considered to be alive, dead in otherwise. So when a cell attains 1-state, it keeps its current rule, on the contrary whenever it becomes dead, it changes its rule. The live/dead state terminology was introduced in the famous artificial life simulation "Game of Life" [13].

Agreement: A cell is in agreement if it is in the same with at least four of its neighbors.

So if a cell is either alive or is in agreement, it is considered to be a successful cell, and its rule remain unchanged. When a cell is unsuccessful, (i.e. dead or in a different state than most of its neighbors) its rule will be changed. Updating its rule is fairly simple, 1. Select a successful neighbor at random, 2. Copy its rule. If there is no successful neighbor, rule will not be changed. A different method is to copy the most successful rule. This copying process may contain a mutation factor. Rule can be copied as it is or it can change during copying. This property introduces mutation concept to rule evolution. Realize that evolution and mutation are some of the fundamental features of life. This is why non-uniform cellular automata concept is devised. Since rule evolution is possible in non-uniform cellular automata framework [14].

Moshe Sipper suggested an enhanced version of NCA with some extra features [14]. According to his enhancements, a cell can be vacant (no rules, i.e. empty cell) or operational (occupied, i.e. containing a finite state automaton). Besides a cell can perform following 4 actions in one time step. Interestingly, his new enhancements mainly focused on the mobility of cells. Remember that mobility is an essential property to mimic life, because most of the living things in nature move.

1. A cell has access to its own state and to its neighbors.

2. A cell can change its own state and its neighbor's state. If more than one cell tries to change the same cell's state, this causes a contention, and this contention is solved

randomly or according to some rule.

3. A cell can copy its rule to a neighboring vacant cell. As you may have noticed, if a cell creates one copy at a time and erase the other cell, it becomes mobile i.e. moving on the grid. Again if more than one cell tries to move to same vacant cell, contention occurs and this can be solved randomly or rule based.

4. A cell can have more than one rule, and only one of them can be active at a time. Moreover these active/passive rules can be copied/moved to other vacant neighbor cells. If contention occurs it can be solved randomly or rule based.

Above properties focused on mobility and self-reproduction. As a result any cell can move on the grid and make copies of self. Doubtlessly a nice example for such a case would be Langton's loop. Evolution progress of Langton's loop can be seen in the below figure [20].

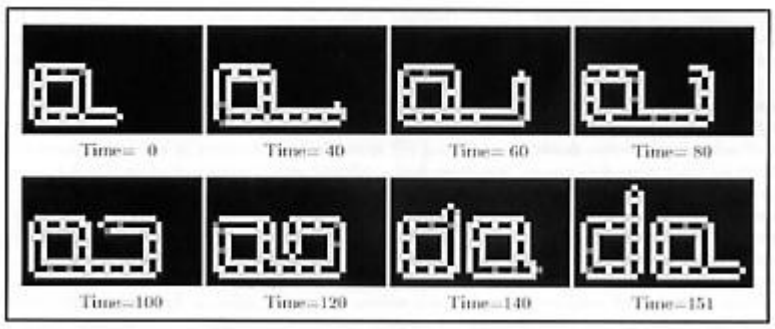
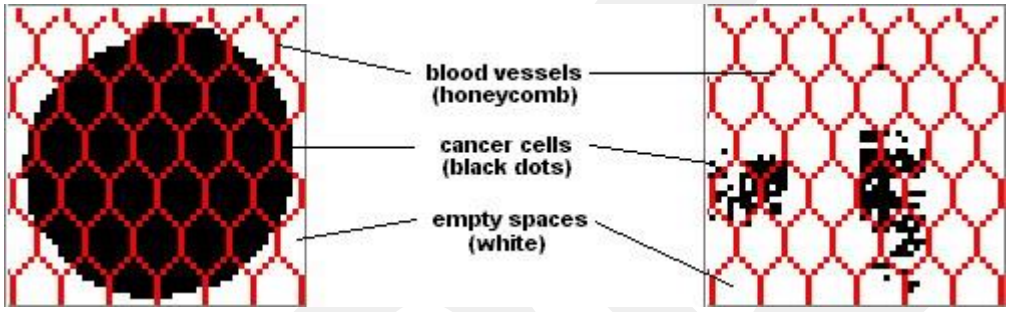


Figure 8 – Langton's loop.

In Figure-8, after some period of time, loop creates an exact copy of itself. If we call this copy-time as cycle, and we continue to run the simulation, after one more cycle there will be 4 loops, so in every cycle, number will be doubled. Self-reproduction has a high importance, and shows the robustness of this environment, plus it is indeed successful in modeling and mimicking living entities. It is obvious that without mobility and self copying, it is harder to achieve such simple design as Langton's loop. For instance this particular example may demonstrate a simulation for growth rate of some bacteria species. Application area of NCA is large as the order of the complexity of its search space. Biology, Chemistry and Computer

Science are major subjects of its usage area. In biology, it is often used to observe tumor behavior and tumor growth rate to increase benefits of cancer therapy [15], in chemistry it is used to observe different material diffusions [16], in computer science it is used to generate random numbers [17], 3-D texturing [18], and cryptology [19]. Below figure shows an example of practical usage of NCA in biology domain. In Figure-9, the square environment contains random pattern NHL (Non-Hodgkin lymphoma) cells (i.e. cancer cells.) and honeycomb-shaped blood vessels.



(a) (b)

Figure 9 – (a) A fully populated cancer cells and (b) cells that suffer from depletion.

By observing such simulation of cancer cell groups, cancer treatments are aware of conditions that stop or partially remove cancer cells. It is appropriate to underline the importance of forward and backward problem in this example. Because without solving forward problem for this example we cannot model the behavior of tumors, and without solving backward problem we cannot identify the underlying rules for known tumor behaviors. Non-Uniform Cellular Automata is a relaxed CA model, the cell neighborhood/connection topology is not same for every cell, and also the transition rules may be different for each cell. So these properties make Non-Uniform Cellular Automata model still a special case of Automata Networks.

2.3 Random Boolean Networks

Let's give simple historical background information for "Random Boolean Networks" (RBN). It was introduced by Kauffman in 1969 and was aimed to be an abstract model for experimenting dynamics of genomic regulatory networks [11].

Each node in the network is symbolizing a gene. Connections between nodes are the regulatory relationship between the genes. Consider 2 nodes, x and y. If there is a connection from x to y, it means that x regulates the output of y. That is, the state of node y at t+1 will be decided according to the state of node x at time t. Basically RBN are another generalized form of Cellular Automata model. In RBN, state of a cell can only be either 1 or 0, nothing else. Unlike, in CA environment, a cell state value can be more than 2, but still finite. Underlying philosophy in RBN, exploits Boolean concept to make a simple abstraction for being active/passive property of genes. Simply, something is either on or off. The robustness of the abstraction is that if we can model everything that fits to this definition (ON or OFF). There is also another difference from CA model. In RBN, neighborhoods need not to be with near cells/nodes. There may be other cells between two far away neighbors (see Figure-10).

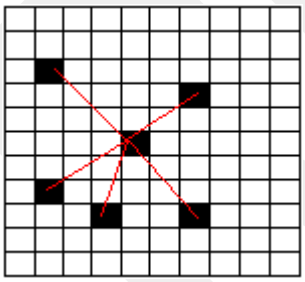


Figure 10 – Nonlocal neighborhood (Irregular)

Also inputs for nodes may differ; one may have three inputs (neighbors) while the other may have 5 inputs (neighbors). The basic idea for this non-homogeneity is to create complex wired structures. So by this way, representation for an intrinsically complex entity becomes easier to define. Figure-11 shows an example RBN topology. Realize that, below nodes can be easily put on a grid, and the distance between the nodes need not to imply the distance between nodes on the grid. It is schematic representation and nodes may be placed on the grid in a more tidy way (visually optimized). Any topology with finite number of 2-state nodes can be thought as an RBN.

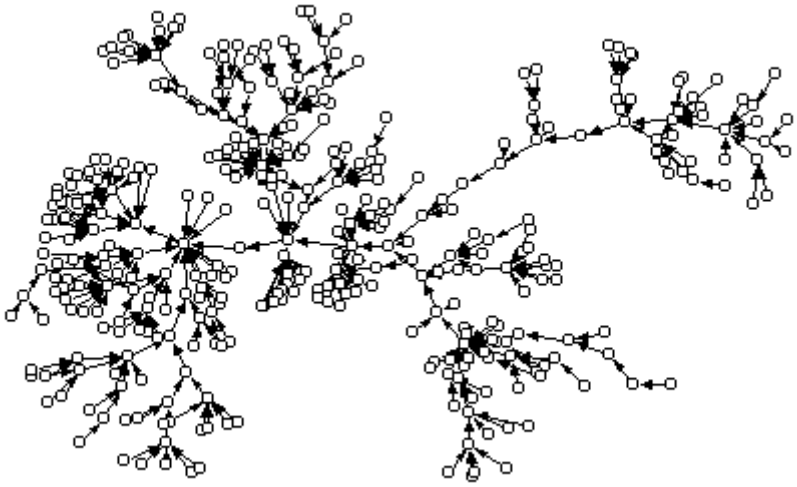


Figure 11 – A RBN topology.

If the number of inputs is the same for all nodes, it is a special case and it is called as Kauffman networks [11]. Each node has its own input to output mapping lookup table. For instance, let's say for RBN, neighborhood is same (say K) for all cells/nodes. For each node in this RBN, there will be possible 2^K number of possible input combinations. If K=3, there will be 8 different input combinations, and for each combination, there will be an output 1 or 0. Below table is an example for a lookup table.

State input from node x, at time=t	**State input from node y, at time=t**	**State input from node z, at time=t**	**Output State for current node n, at time=t+1**
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Table 2 – An example rule lookup table, (Values in the output column set arbitrarily).

Consider a RBN with finite number of nodes, so it has to cycle from some point, because the input-output mapping has to be deterministic and there are finite numbers of configuration. So, after all configurations are occurred, RBN becomes to cycle, so it is periodic. In dynamical systems terminology this is called attractor [11]. Consequently, because of the above mentioned properties, RBN are also considered to be a generalized version of Cellular Automata.

2.4 Generative Network Automata

When we consider previously discussed frameworks, they are all limited with one strict property. The system is assumed to have a fixed topology and the dynamics of the system consists of only state changes of the nodes. At this point "Generative Network Automata" model which is proposed by Hiroki Sayama and Craig Laramee, aimed to disperse this deficiency [1]. Importantly in GNA perspective, an outside effect (e.g. perturbation) may be a reason for why the network structure changes its topology by adding, removing and modifying its nodes. So, the structure of the system may open to external effects and this makes the model one step closer to real life like entity. If we consider a real life complex network like internet, states and topology changes occur simultaneously. This is called "coevolving" in terms of GNA. Some people join the internet topology while others leave or change their state on the network (busy, available etc...). These events occur concurrently and this makes it harder to model in usual complex dynamic system frameworks. This issue can be addressed in GNA modeling framework via the concept of graph rewriting. Briefly, the methodology utilizes Dynamical systems theory (GNA is a discrete dynamic system modeling tool), Network theory (varying topologies) and Graph theory (graph rewriting by extraction and replacement mechanisms). Table-3 gives some examples to these discussed real world complex networks.

Network Type	Nodes	Links	States of nodes	Node addition/removal	Topology change
Biological organisms	Cells	Cellular communications	Gene activities	Cell division / death	Migration
Ecological community	Species	Ecological actions, predation, symbiosis	Species diversities	Invasion, extinction, spread.	Adaptation, crossbred etc…
Disease-spread network	Individuals	Physical interaction	Pathologic state.	Death, quarantine	Lack of physical contacts
Social network	Individuals	Cooperation, Competition	Social position. (leader, wealth)	Acceptance or outcast from the community	Establishment of relationships.

Table 3 – Some real life examples with concurrent state and topology change due to network dynamics.

A formal definition for Generative Network Automata model is given by its authors [1], which follows as below;

V_t : Finite set of nodes, which changes over time by addition/deletion of nodes from the network.

C_t : is a mapping that is $V_t \rightarrow S$ from node set to node state set S. It can vary over time.

L_t : $V_t \rightarrow \{V_t \times S\}^*$ A mapping from node set to list of destinations of outgoing connections and the states of these connections. It represents the global topology of the network at time t.

Graph rewriting procedure occurs through three basic operations. A formal definition of GNA dynamics is defined by a triplet of < E, R, I >. Below, we explain these operations:

(E)xtraction: This mechanism selects which part of the topology will be updated, as a function, it takes the whole topology as a parameter and returns a sub-part of it, this selection mechanism may be based on a rule or it may be random.

(R)eplacement: The replacement operates as a function, it takes a sub-GNA (sub-part) which is previously selected by Extraction mechanism and returns a pair. Pair consists of a new sub-GNA and a correspondence between new sub-GNA and the old one. Underlying rule/mechanism may be based on a rule or it may be random.

(I)nitial: Initialize/create a new configuration of GNA.

Obviously above basic operations have some other not-detailed operations like, removal of nodes, links between nodes. But these details are design specific and may be implemented within these operations. They are not given because their definitions are not crucial in the aspect of this methodology and fairly simple. In Figures 12-14, the basic operations are explained.

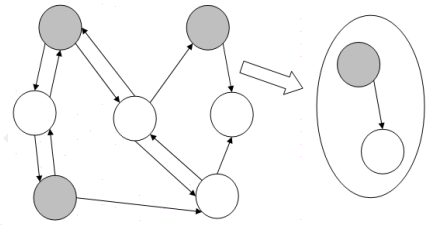


Figure 12 – Extraction mechanism selects a sub-GNA and returns it.

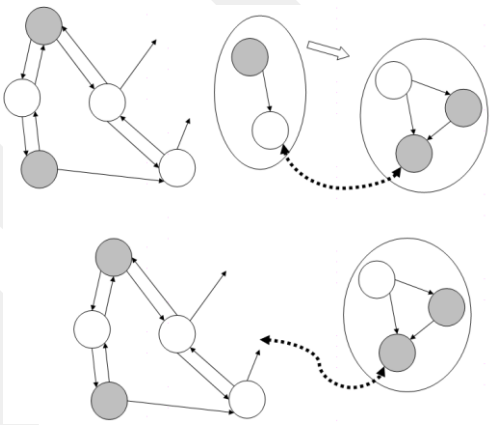


Figure 13 – Evolving of the sub-GNA and replacement.

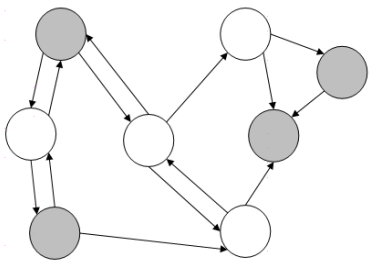


Figure 14 – Resultant GNA

The figures briefly depict the GNA rewriting process. The generality of GNA modeling framework is valuable. Since all other discrete dynamical system modeling environments can be modeled and realized within GNA framework, it can be thought as the father model of all others models like; ordinary Cellular Automata, Random Boolean Networks, Hybrid Cellular Automata, Artificial Neural Networks and etc… When Automata Networks and GNA are compared, it is obvious that GNA allows changing topologies whereas Automata Networks do not. However we may encode a GNA model and then it can be represented in Automata Network model and also vice versa is also possible, an Automata Network model can be encoded and represented in GNA framework. One of the most important novelties introduced by GNA is changing topology of the network as discussed and shown above. So, being able to change topology, we can model many (most) real world complex discrete networks. Of course, there are computational limits when we are talking about systems with more than 1000 nodes (many of the real world examples have more than that). These computational limits may be pushed further, when better extraction and replacement mechanisms are applied. These are all representational and practical limits. The theory behind this framework is robust and promising. Of course the framework is going to be better with improvements made on representation issues, extraction and replacement mechanism or any other improvements that exploits computational issues.

CHAPTER III

GENERATIVE NETWORK AUTOMATA IDENTIFICATION

In complex dynamical systems domain, the study of behavior is realized in two main approaches. The first one is to start with global information describing state and topology changes of the system and identify the local state transition/topology transformation functions that generate them (i.e. the *inverse* problem). The second approach is to start with some state transition/topology transformation functions at hand and observe the resulting behavior of the constructed system through local function execution (i.e. the *forward* problem) [4,5,6].

This thesis has focused on the first approach, the inverse problem. The environment is the mentioned GNA modeling framework, which is known to be a varying-topology extended modeling framework for complex dynamical systems [1]. GNA framework provides us a well-defined environment for examining potential relations and mechanisms between part and the whole. Before giving algorithm details, definitions and assumptions are given first. (These definitions and assumptions are already given in a prior study [12]).

3.1 Definitions and Assumptions

GNA provides a dynamically changing network description via a time-varying directed graph with its labeled nodes. Each node of the graph represents one system component and takes one of the possible state values defined by a state value set S. The edges are supposed to be the indicators of referential relationships between system components that cause state transitions and topology transformations in time.

Below, we give definitions about GNA configuration and its temporal dynamics [1].

Definition 1: *A configuration of a GNA at time t* is a triplet $\langle V_t, C_t, L_t \rangle$ where

V_t : a finite set of nodes at time *t* describing dimension of the system,

C_t : a mapping $V_t \to S$ defining the global state of the system at time *t*,

L_t : a mapping $V_t \to V_t^*$ defining global topology of the system at time *t*. This mapping is defined by outgoing edges of a specific node to its destination nodes.

Definition 2: Let G be the set of all GNAs. *Temporal dynamics of a GNA* is a defined by a triplet $\langle E, R, I \rangle$ where

E : a mechanism that produces an extraction $g_e \in G$ from a given GNA instance $g \in G$.

R : a mechanism that replaces the extracted subGNA g_e with a new GNA instance $g_n \in G$ and produces correspondence $V_t^{g_e} \to V_t^{g_n}$ from nodes of g_e to nodes of g_n.

I : an initial configuration of GNA.

The extraction and replacement mechanisms can be deterministic or stochastic. The above extraction mechanism E, replacement mechanism R and initial configuration I are sufficient to define specific GNA models and reflect a specific perspective (defined by $\langle E, R, I \rangle$ structure) to explain complex dynamic behavior of systems under study. Clearly, based on the amount of information provided by given global behavior data, there may be more than one GNA model to be identified by an identification algorithm. This fact has also been pointed out in the context of reverse engineering of Automata Networks [8]. So, one needs to define his/her assumptions about the characteristics of the model(s) under study, for a precise identification. The assumptions/restrictions can be treated as a priori knowledge about the system under study that may vary from biological organisms, ecological communities and human societies to computer communication networks. Our assumptions include the following:

- Local behavior of system components are supposed to be *deterministic*. As a consequence, whole system behavior is supposed to be deterministic which is reflected by global behavior data under study.
- Components have *minimal memory*. In their update, components consider only one step previous state/connectivity of other system components (this may also include themselves).
- System evolves *synchronously* in terms of its component states and network connectivity updates.
- Global behavior of the system is supposed to be *cyclic*. In other words, the data reflecting last configuration is supposed to be followed by the first one.
- We divide system into two parts: *observable* part and *reservoir* part (see Figure 15). Whole system is supposed to be *isolated* and its components are *distinguishable*. However, the observable part of the system from which global behavior data input is extracted is *non-isolated* and its components are still *distinguishable*.
- Components are supposed to appear at least once in the observable part of the system during their lifetime.
- All systems components are assumed to be known a priori in the form of a finite ordered set H defining inter-component neighborhood relation based on ordering.
- Component behaviors are supposed to be explainable by state/connectivity information fed from *nearest neighbor* components.
- At any time a component can be in one of three states represented by 0, 1 or 2 symbols (i.e. S = {0, 1, 2}). A component residing in the observable part of the system may take state value either 0 or 1. On the other hand, the state value of a component in reservoir part can only be 2. This value indicates unknown states of separate/disconnected components.

An example snapshot of a system is given in Figure-15 to clarify the sets and the notations used in the definitions and assumptions. From Figure-15 we can conclude that set of system components H = {p, q, r, s, t, u, v, w, x, y, z}; finite set of components at time *t*, V_t = {p, r, t, v, w, y, z}; component states $C_t(r) = C_t(t) = C_t(v) = C_t(z) =$ 0 and $C_t(p) = C_t(w) = C_t(y) =$ 1; destination

component sets for each component $L_t(p) = \{z\}$, $L_t(r) = \{v\}$, $L_t(t) = \{p\}$, $L_t(v) = \{p, w\}$, $L_t(w) = \varnothing$, $L_t(y) = \{r, t, w\}$, $L_t(z) = \{t, y\}$. The state of any component at reservoir part is unknown and symbolized by 2.

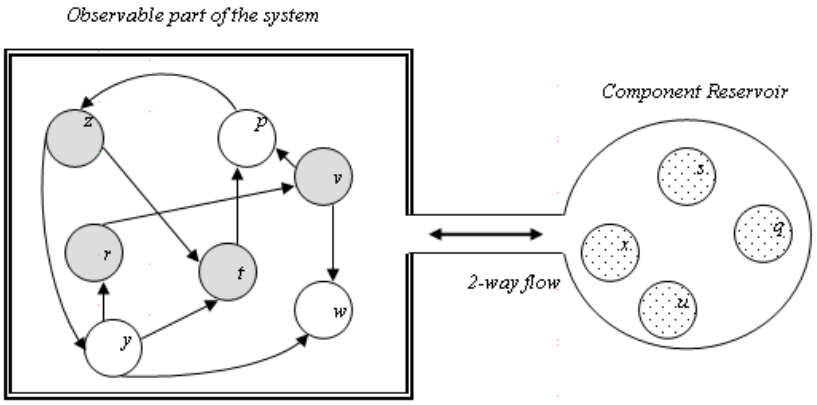


Figure 15 - An example system snapshot.

3.2 Algorithm

In this section, we define the identification algorithm for the restricted type of systems defined by the assumptions given in section 3.1. The inputs of the algorithm are a sequence of GNA configurations reflecting given system's global behavior, a finite ordered set H of components from which V_t sets are constructed and state/connectivity interaction mode considered in the identification process.

A fundamental question that we have to answer is: "How should component states and connectivity affect each other so that given autonomously varying topology and state change sequence can be generated?". For that purpose, the algorithm considers three alternative state/connectivity interaction modes which characterize single component behavior that drive general GNA system evolution: *reader* mode, *writer* mode and *transfer* mode. For a given system, all components are supposed to behave in the same mode. In reader mode, component i is first supposed to be able to identify its incoming edges and their corresponding source nodes $P_{t-1}(i)$ (this set may also include the component itself) together with their states $C_{t-1}(j)$ where $j \in P_{t-1}(i)$ then autonomously decide on new incoming components $P_t(i)$ and the component i's new state $C_t(i)$ (see Figure-16).

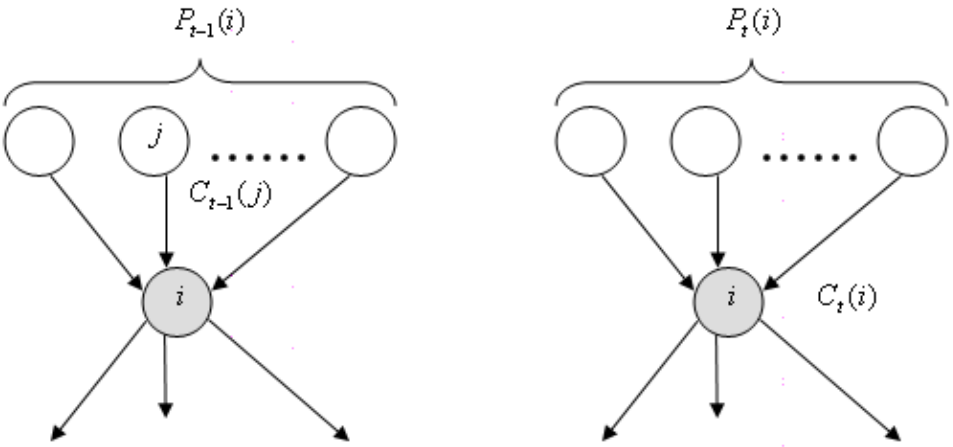


Figure 16 - Component *i* working in *reader* mode, whose inputs are component set $P_{t-1}(i)$ and their $C_{t-1}(j)$ values, outputs are component set $P_t(i)$ and $C_t(i)$ value.

Similarly in writer mode, a component *i* first determines destination components $L_{t-1}(i)$ through outgoing edges and their corresponding states $C_{t-1}(j)$ where $j \in L_{t-1}(i)$ then the state and connectivity information is used to generate new outgoing components set $L_t(i)$ and new state of component *i* $C_t(i)$ (see Figure-17).

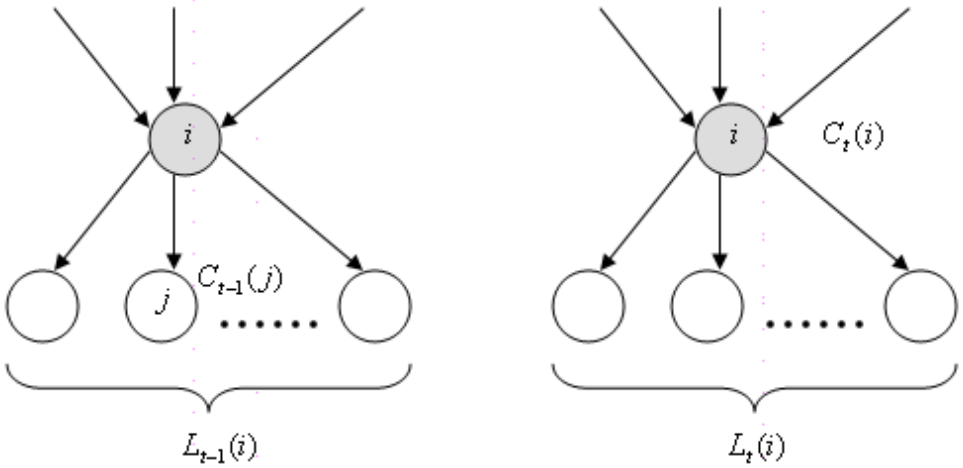


Figure 17 - Component *i* working in *writer* mode, whose inputs are component set $L_{t-1}(i)$ and their $C_{t-1}(j)$ values, outputs are component set $L_t(i)$ and $C_t(i)$ value.

In transfer mode, component *i* is supposed to generate new outgoing edges (i.e. destination components $L_t(i)$) and its state $C_t(i)$ based on its incoming components

$P_{t-1}(i)$ and their states $C_{t-1}(j)$ where $j \in P_{t-1}(i)$ of the corresponding source components in previous step as shown in Figure-18. In this mode, the effect of individual component *i* over the whole system behavior is realized through transformation of incoming connectivity and corresponding state information to outgoing components and the component's state.

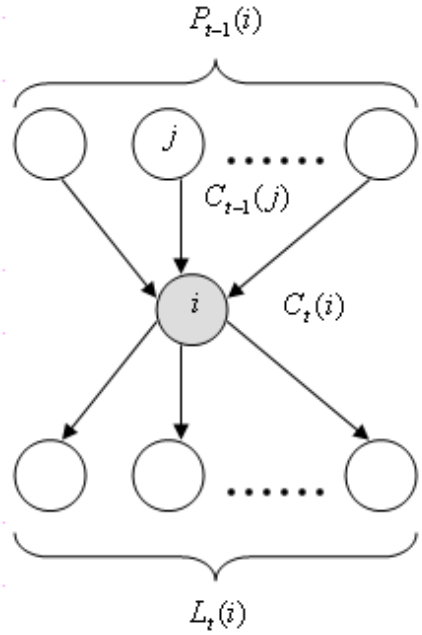


Figure 18 - Component *i* working in *transfer* mode, whose inputs are component set $P_{t-1}(i)$ and their $C_{t-1}(j)$ values, outputs are $L_t(i)$ and $C_t(i)$ value.

In computations, P_t is assumed to be a mapping $V_t \to V_t^*$ defining global topology of the system at time *t* by incoming edges of node *i* from its source nodes. We can relate P_t and L_t in such a way that for all $j \in P_t(i)$ we require $i \in L_t(j)$ and for all $j \in L_t(i)$ we require $i \in P_t(j)$. Clearly, one can obtain all P_t values from all L_t values. None of the proposed modes cause conflicts in terms of their decided state and connectivity values. One can consider alternative component behavior modes that can explain the restricted systems' global behavior in terms of (connectivity, state) input tuple generating (connectivity) and (state) outputs (see Table-4).

	INPUT		OUTPUT		
MODE ID	**(Connectivity, State)** $(P_{t-1}(i), C_{t-1}(j))$	**(Connectivity, State)** $(L_{t-1}(i), C_{t-1}(j))$	**(Connectivity), (State)** $(P_t(i)), (C_t(i))$	**(Connectivity), (State)** $(L_t(i)), (C_t(i))$	**COMMENT**
1	0[a]	0	0	0	NU[c]
2	0	0	0	1[b]	NU
3	0	0	1	0	NU
4	0	0	1	1	Conflict
5	0	1	0	0	NU
6	0	1	0	1	*Writer*
7	0	1	1	0	NC[d]
8	0	1	1	1	Conflict
9	1	0	0	0	NU
10	1	0	0	1	*Transfer*
11	1	0	1	0	*Reader*
12	1	0	1	1	Conflict
13	1	1	0	0	NU
14	1	1	0	1	NC
15	1	1	1	0	NC
16	1	1	1	1	Conflict

[a] 0 means Input/Output item is not used in the mode, [b] 1 means Input/Output item is used in the mode,
[c] NU : Mode has No practical Usage, [d] NC : Mode that is Not Considered.

Table 4 - List of possible [(Connectivity, State)] → [(Connectivity), (State)] based component behavior modes for minimal memory restricted systems under study.

In Table 4, a component behaving in modes 1 to 4 does not get any input from the others. In other words, system components are disconnected. They have no meaning in GNA context. Similarly, modes 1, 5, 9 and 13 do not produce any output so they do not define a dynamical system and have no practical importance. Component behavior modes 4, 8, 12 and 16 update both incoming and outgoing components' connectivity, simultaneously that results in conflicts. Therefore, we did not consider these modes during the identification process. Finally, modes 7, 14 and 15 are not considered but they can easily be added to the implementation.

In the implementation of the algorithm, information about existence of incoming or outgoing connectivity between components (defined by sets $P_t(i)$ and $L_t(i)$) is encoded as 0 and 1 for simplicity. The following examples explain the encoding used in the algorithm.

Example 1.

Consider a simple configuration sequence with a single node i (i.e. H = {i}) working in *reader* mode as indicated in Figure-19.a. The algorithm identifies an encoded coupled state/connectivity transition rule shown in Figure-19.b.

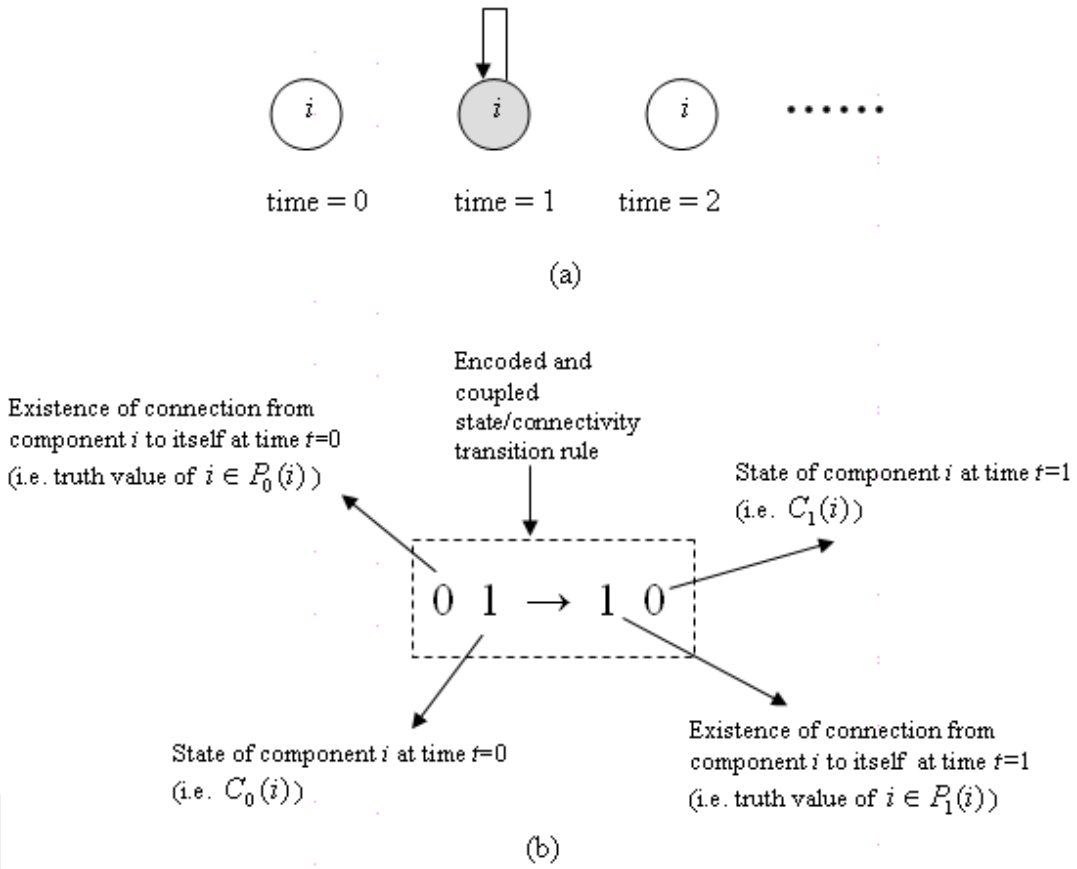


Figure 19 - A simple configuration sequence (a) and an extracted state/connectivity transition rule for the system working in *reader* mode (b).

Depending on given configuration sequence, deterministic behavior of an individual component may not be explainable only by its own history. We call it *inconsistency*. They are resolved by neighborhood expansion of component in concern. So, the inconsistencies are resolved by the establishment of what we call *overlay* level component dependency settings that leads to the identification of an overlay level network topology. Initially, each component is assumed to depend only on itself. Therefore, the only overlay level neighbors of components are themselves. Any

neighborhood expansion may continue at most until all system components are added into the current overlay level neighborhood of the component under construction.

Note that any inconsistency is guaranteed to be resolved at worst when the identified overlay network become fully-connected. This is because the given global behavior of the system has been assumed to be deterministic. The following example covers an inconsistency situation resolved through component overlay neighborhood expansion realized over finite ordered set H of system components given as input.

Example 2.

Consider a simple configuration sequence with two components *i* and *j* depicted in Figure-20.a and H = {i, j}. The system is assumed to work in *reader* mode. The algorithm identifies an inconsistency case, see Figure-20.b, then it generates an encoded coupled state/connectivity transition rules for component *i* as shown in Figure-21.c through overlay level neighborhood expansion. Note that pattern "**0 2**" in Figure-21.c is constructible over component *j* since we know that $j \in H$ and *j* is the next neighbor of *i*, a priori. The result of the neighborhood expansion is the establishment of overlay level dependency of component *i* to *j* (see Figure-21.d). The sequence does not result in any inconsistency for component *j*, so its overlay level neighborhood remains limited to itself, only.

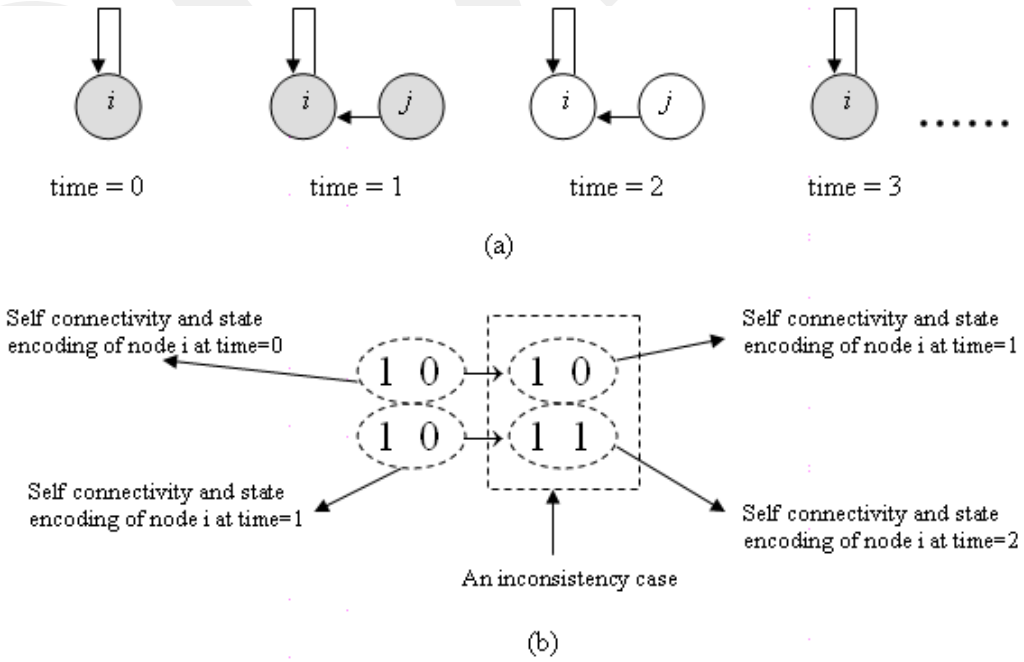


Figure 20 - A configuration sequence (a), an inconsistency case (b).

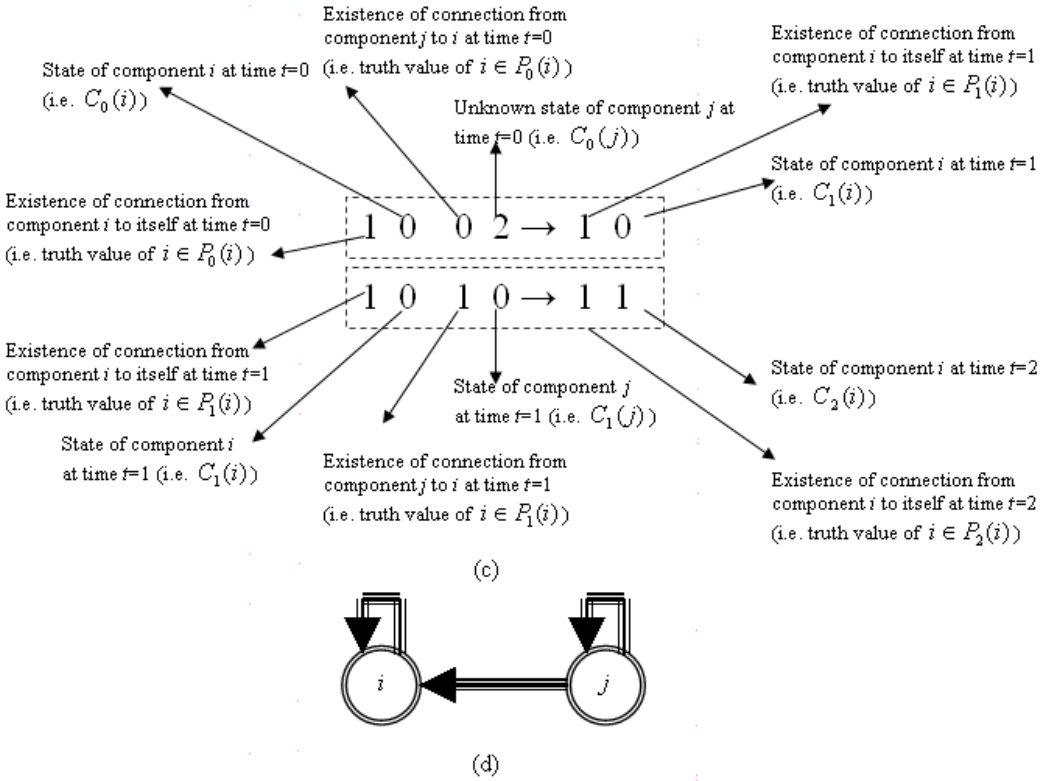


Figure 21 - Extracted state/connectivity transition rules through nearest neighbor overlay expansions (c) and the identified component connectivity (d) for the system working in *reader* mode.

Inputs:

- a deterministic sequence of k GNA configurations $\langle V_i, C_i, L_i \rangle$ labeled from S_0 to S_k where $0 \le i < k$ and $S_0 = S_k$,
- a finite ordered set H of system components such that $V_i \subseteq H$ for all $0 \le i < k$,
- State/connectivity interaction mode read/write/transfer.

Outputs:

- An identified set of encoded and coupled state/connectivity transition rules T_i for each system component $i \in H$.
- Overlay level neighborhood graph G describing the flow of state/connectivity information between system components.

Note that the algorithm (see Appendix A for pseudo code, and also implemented C code can be found in compact disc which is included in the back of this book.) does not directly produce the $\langle E, R, I \rangle$ triplet describing temporal dynamics of GNA but instead transition rules that describe component level local behavior together with overlay level neighborhood topology are the outputs. In fact, the outputs describe nothing but an Automata Network. Left-hand-sides of the identified AN transition rules describe the parts to be extracted from GNA due to a single component subGNA and right-hand-sides define the parts that replace the extracted left-hand-sides on the same single component subGNA scale. The AN transition rules are sufficient to generate given global system behavior because it can be explained by collective behavior of individual system components co-evolving through state and connectivity based overlay level complex neighborhood interactions. Depending on the nature of the sequence, the identified AN may fit to instances of cellular automata, random Boolean networks, graph grammar, pure network growth models or any other hybrid GNA model. Finally, the identified AN model might be partially defined since the information provided by GNA sequence is limited to some finite number of configurations.

3.2 Algorithm worst-case and best-case analysis

In worst-case scenario;

We expect a conflict will occur for every component in the last row, and the neighborhood will be expanded up to maximum number (which is the total number of components n).

```
1 For each component i
2 {
3 Set rule set T_i to ∅;
4 Set component i's overlay level neighbor set G_i to { i };
5 }
```

The above code block will run at most ***n*** times for a system with ***n*** components. Let the cost for the line 3 be T_1 and cost for the line 4 be T_2.

Similarly, the following code block will run ***n*** times and the cost for the line 9 be T_3.

```
6 For each component i
7 {
8  If (mode = reader or mode = transfer)
9    Construct the sets P_t(i) by using L_t(j) of other components
10   in given sequence;
11 }
```

Up to now, total cost for the lines 3,4 and 9 is $n.T_1 + n.T_2 + n.T_3$.

For the rest of the algorithm, let the total cost for the lines between 14-38 (inside of the for loop) be T_4 and let *k* be the number of rows (total number of time steps for configuration history). The total cost for the following block is $((n^3 + n^2)/2).k.T_4$.

```
12 For each component i
13 {
14  t = 0;
15  While t < k do
16  {
17   Switch (mode)
18   {
19    case reader : Construct next rule u by
20    using P_t(i) , P_{t+1}(i) , C_t(j) and current G_i ;
21    case writer  : Construct next rule u by
22    using L_t(i) , L_{t+1}(i) , C_t(j) and current G_i ;
23    case transfer : Construct next rule u by
24    using P_t(i) , L_{t+1}(i) , C_t(j) and current G_i ;
25   }
26   If (u is consistent with all the rules in T_i)
27   {
28    If (u ∉ T_i) Set T_i = T_i ∪ {u} ;
29    t = t + 1;
30   }
31   Else
32   {
```

```
33    Set rule set T_i to ∅;
34    Find next overlay level nearest neighbor
35    component n by using ordered set H;
36    Set G_i = G_i ∪ {n};
37    t = 0;
38    Start over while loop;
39   }
40  }
41 }
```

Let the cost for the last line be T_5 and it will run at most $n.T_5$.

```
42 Return all sets T_i and graph G;
```

Total sum is $n.T_1 + n.T_2 + n.T_3 + ((n^3 + n^2)/2).k.T_4 + nT_5 = \mathrm{O}(n^3 k)$

In best-case scenario;

Only difference occurs in the lines 12-41, we expect the algorithm find the rules for components in single pass, so the lines will run in $n.k.T_4$

The sum is $n.T_1 + n.T_2 + n.T_3 + n.k.T_4 + nT_5 = \Omega(nk)$

The assumptions made in section 3.2 affected algorithm's run time. It reduced the size of the search space, since the identified system is more specific and restricted with assumptions. As a consequence, in both scenarios, the algorithm runs in polynomial time.

GCRIS

CHAPTER IV

EXAMPLE APPLICATION: SIMPLE LEGO® DESIGN

All real world ‘situations’ are mainly based on changes. Consider a robot walking on a street environment; it is a state-changing entity. The state space is vastly big; due to the fact that world itself is changing every passing moment. These changes can be expressed by translating them to mathematical representations, and then they can be modeled and studied in one of the discrete dynamic system modeling environments.

In this chapter, an example from Lego®-based design identification domain is introduced. It is selected due to its simple discrete nature and implied high level of abstraction. It provides us a simple environment for engineering design identification problem. The identification process itself can be considered as the first step of possible automated design effort. For any design domain, the design process involves selecting *right* components and *attaching* them to other components correctly. The words ‘*right’* component and `*attaching*` can be thought as analogous to component *state* and component *connectivity* in GNA context. The lifetime of a design process can be defined as discrete encodings that are developed over discrete time steps. At the beginning of a design process, there must be at least one component. After each time step, new components are added/deleted and new connections are established/cut until the design meets its requirements. Throughout the process, conflict cases, discussed in section 3.2 may occur. Such cases can be resolved with nearest neighborhood descriptions and the assumptions as mentioned before. Alternatively, we may prefer `User-Guidance` for conflict resolving.

During the design process, our designer may encounter with a conflict case and he/she solves this conflict case with the help of his/her previous experiences. These

experiences are replacing components and/or establishing new connectivity among components according to designer's point of view. In order to make use of these experiences, we can prepare a look-up table which tells us how to extend the component neighborhood in conflict cases.

Components	**1. Neighbor**	**2. Neighbor**	**3. Neighbor**	**4. Neighbor**	**5. Neighbor**
a	b	e	d	c	f
b	d	f	a	e	c
c	f	a	d	e	b
d	b	a	c	f	e
e	f	b	c	d	a
f	a	c	d	b	e

Table 5 – User-guidance table for conflict resolution.

In Table-5, if a conflict occur, for component **a** then we must use component **b** to explain the conflicting behavior of component **a**. if another conflict occurs, use component **b** and **e** (i.e. expand current neighborhood) to explain the conflicting behavior of component **a**. Briefly, we are expanding the neighborhood of component **a**, every time a conflict occurs. Rows of the above table give us each component's conflict resolution scheme (neighborhood expansion scheme). Remember that earlier, in the nearest neighborhood approach, we were limited to use the nearest neighbor node to explain the behavior, but in this heuristic, relationships become more complex. Another important case is, this lookup table is prepared by the designer, in forward problem aspect, he/she establishes the connectivity according to this table but in the inverse problem aspect, we try to explain the conflicting case by looking at this table. So, if the look-up table could have been given to the algorithm, the output of the algorithm (identified-rules) would be the design simple rules reflecting the designer's way of thinking. At this point, we need to modify the ID_GNA algorithm in order to incorporate the user-guidance for conflict resolution.

Modified part of GNA Identification Algorithm:

```
…
31   Else
32   {
33    Set rule set T_i to ∅;
```

```
34    If end-of lookup table reached
35      Set G_i = G_i ∪ {v};
36    Else
37    {
38      Find next overlay level neighbor
39      component n by using lookup table T;
40      Set G_i = G_i ∪ {n};
41    }
42    t = 0;
43    Start over while loop;
44  }
```

…

In lines 34 and 41, we use lookup table to determine the next overlay level neighbor instead of using nearest neighborhood heuristic.

Before giving detailed example, Figure-22 displays three different types of Lego®-bricks under consideration. The algorithm is supposed to work on reader mode for this example.

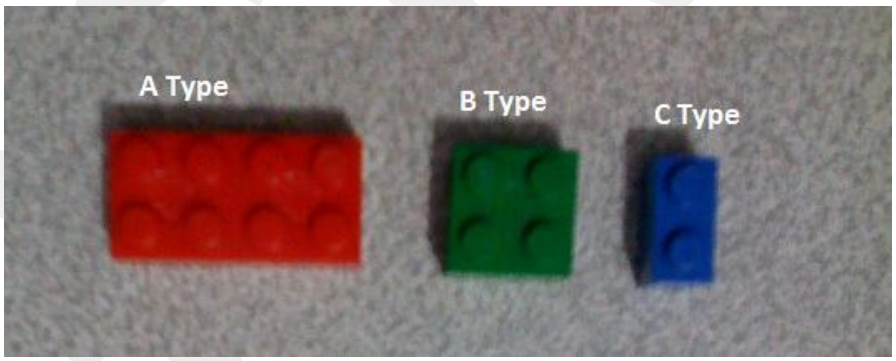


Figure 22 – Three Types Lego® Bricks.

For applicability, these types will be our state value set *S*.

$$S = \{A, B, C\}$$

Consider the following final Lego® bridge design in Figure-23, which consists of eight components.

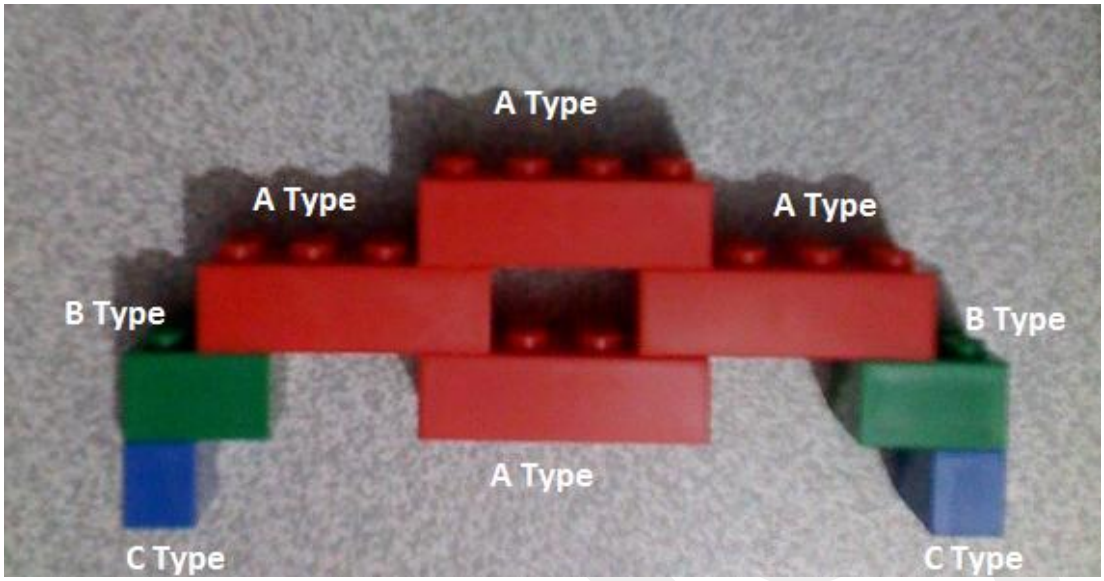


Figure 23 – Lego® bridge design.

So our finite ordered node set H contains 8 nodes/components.

$$H = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

Build process for the bridge is illustrated from Figure 24 to Figure 28 and the each step is expressed in terms of GNA notation, so later they are given input for the algorithm. (At t=0, there is no node in the system, first step starts t=1).

First Step t=1, we have only Component 1 in the observable part, other components are in reservoir.

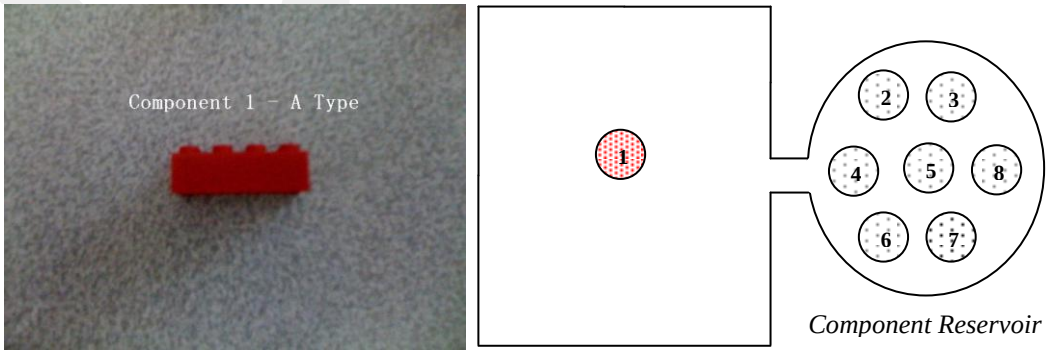


Figure 24 – Configuration of the system at time t=1.

Second Step t=2, components 2 and 3 are attached to component 1.

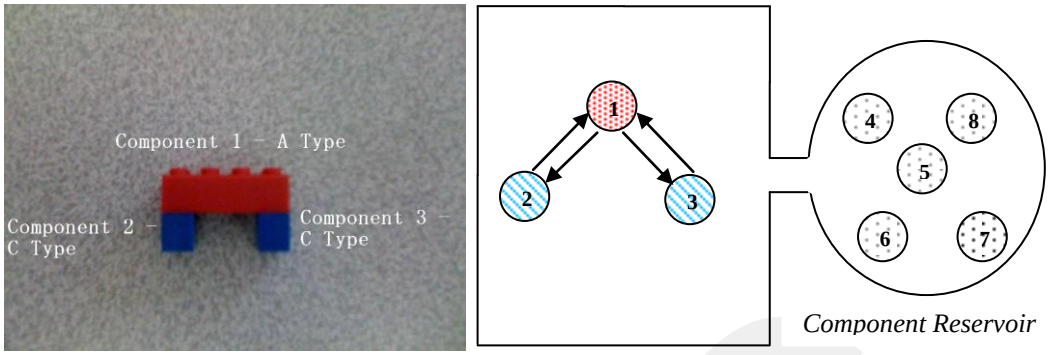


Figure 25 – Configuration of the system at time t=2 .

Third Step t=3, components 2 and 3 are detached from component 1 and component 2 is attached to component 4, component 3 is attached to component 5. Components 4 and 5 are attached to component 1.

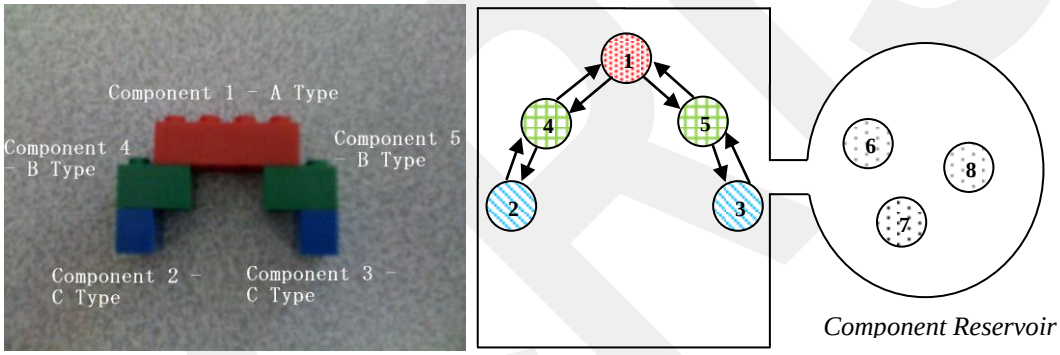


Figure 26 – Configuration of the system at time t=3 .

Fourth Step t=4, similar to the third step, components 4 and 5 are detached from component 1, components 6 and 7 are attached to component 1. Components 4 is attached to component 6 and component 5 is attached to component 7. Component 2 and 3 remains same.

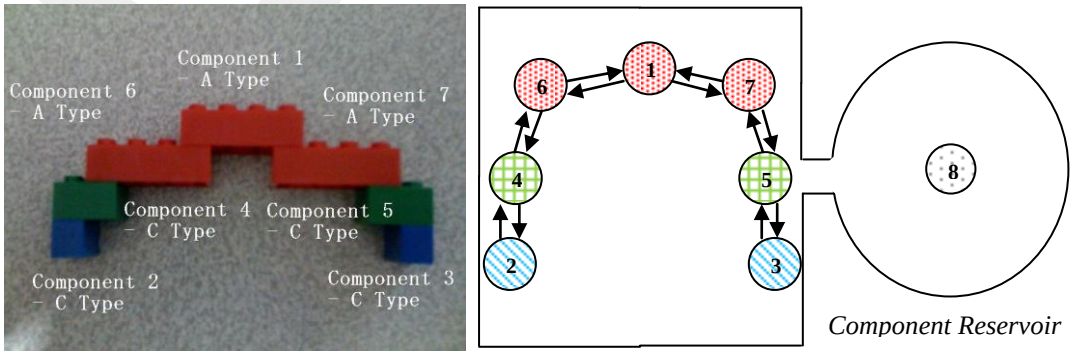


Figure 27 – Configuration of the system at time t=4 .

Fifth Step t=5, in this final step, only component 8 joins the system, and it is attached to component 6 and 7.

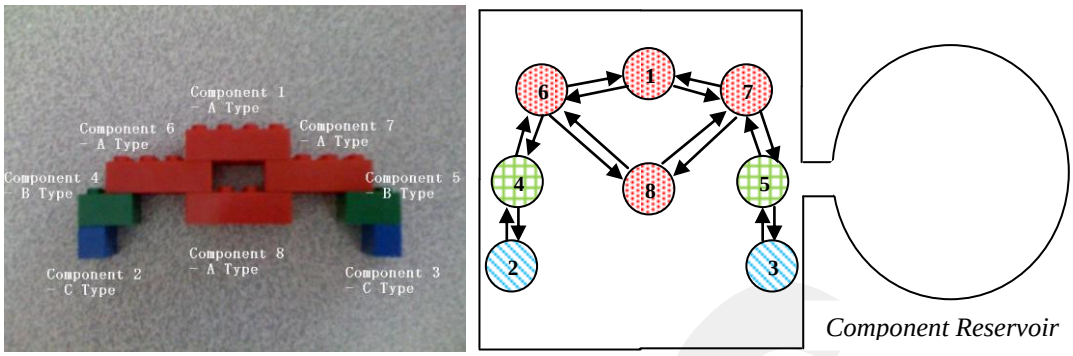


Figure 28 – Configuration of the system at time t=5.

In figure 28, we see the final system at t=5, and from the first step to fifth step it is encoded as follows for component 1's perspective from t=1 to t=5;

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
t=1	00	03	03	03	03	03	03	03
t=2	00	11	11	03	03	03	03	03
t=3	00	01	01	12	12	03	03	03
t=4	00	01	01	02	02	10	10	03
t=5	00	01	01	02	02	10	10	00

Table 6 – From first step to fifth step for component 1's perspective

Each row of the table represents the configuration of the system from component 1's perspective at different time steps, topmost row is the t=1 and the bottom is the t=5. Each column represents connectivity and state pair for component 1's perspective. For instance, look at the row t=4. 1. Column says that component 1 has no connection to itself and its own state is 0 (Type A), 2.Column says that component 2 has no connection to component 1 and its state is 1 (Type C), 3. Column says that component 3 has no connection to component 1 and its state is 1 (Type C), 4. Column says that component 4 has no connection to component 1 and its state is 2 (Type B), 5. Column says that component 5 has no connection to component 1 and its state is 2 (Type B), 6. Column says that component 6 has connected to component 1 and its state is 0 (Type A), 7. Column says that component 7 has

connected to component 1 and its state is 0 (Type A), 8.Column says that component 8 has no connection to component 1, because its state is 3 (Not-joined to system yet, it is in reservoir). Eventually we have defined the triplet $\langle V_t, C_t, L_t \rangle$, since the table gives the finite state of nodes H={1, 2, 3, 4, 5, 6, 7, 8}, C_t states of the nodes at time steps, L_t the topology of the system at different time steps for component 1's perspective. Also we may prepare the same table for other components, but in this example we are interested in component 1. So the algorithm will be run for component 1's perspective. The below table is the guidance matrix for this bridge example.

Components	**1. Neighbor**	**2. Neighbor**	**3. Neighbor**	**4. Neighbor**	**5. Neighbor**
1	6	7	X	X	X
2	4	X	X	X	X
3	5	X	X	X	X
4	2	6	X	X	X
5	3	7	X	X	X
6	4	1	8	X	X
7	5	1	8	X	X
8	6	7	X	X	X

Table 7 – User-guidance table for Lego® bridge example, 'X' represents a Don't Care case.

The table 6 and 7 are represented as 2-d arrays in implementation and given as input parameters and the output (identified rules) is as follows;

Comp.	1	2	3	4	5	6	7	8
Rule 1	00→00	03→11	03→11	03→03	03→03	03→03	03→03	03→03
Rule 2		11→01	11→01	0311 →12	0311 →12	0312 →10	0312 →10	0310 →00
Rule 3		01→01	01→01	12→02	12→02	10→10	10→10	
Rule 4				02→02	02→02			

Table 8 – Identified rules for component 1's perspective.

And see Figure-29 for the overlay level neighborhood graph.

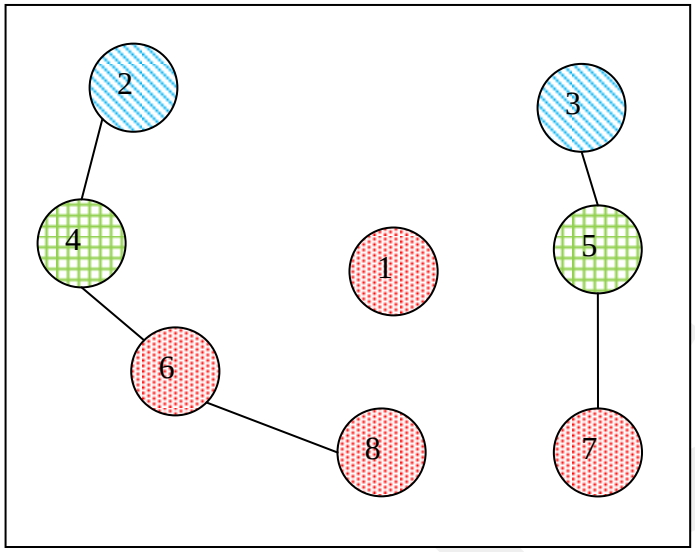


Figure 29 – Overlay level neighborhood graph for Lego® Bridge Example

The overlay level neighborhood graph shows a clear picture of how the inconsistency is solved. Simply this methodology can be applied to other components as well. With identified rules and the configuration at time t=1, we can produce the systems final configuration at time t=5. Briefly, identified rules can used to solve the forward problem (i.e. obtaining system configuration at time t=5).

In this scenario, we did not encounter a case where we reach the end of lookup table and still did not solve the conflicting case. In the modified algorithm pseudo code (page 32), lines 34 and 35 of modified shows the use of a virtual element (*v* in pseudo code stands for the virtual element) in order to solve the conflicting case. In this way, the algorithm does not stop working and continue identifying rules. This virtual element does not have a physical meaning between Lego® parts. It is used to prevent the algorithm to halt. From Table-8, it is clear that, there are conflict cases for components 4,5,6,7 and 8. These conflict cases are resolved by extending neighborhoods for these components according to User-Guidance Lookup Table (see Table-7), so this table has a direct effect on the solution of the problem, and it may increase or decrease solving time.

CHAPTER V

CONCLUSIONS

Identification of the relation between local and global, subsystem and system or simply the part and the whole is a known fundamental question. In this thesis an algorithm to identify the behavior of complex discrete dynamical systems from their global behavior data is developed. The algorithm is applied to a Lego® example for demonstration purposes and local design rules are identified. As shown in the example application, algorithm is practical, in some cases like a system with lots of components (for example 1000 components,) the rule space may be as large as 2^{1000} rules which means that increased computation time. However, nearest neighborhood and user-guidance for neighborhood expansion assumptions make the algorithm to execute in polynomial-time. The worst-case and best-case time complexities of the proposed algorithm are $O(n^3k)$ and $\Omega(nk)$, respectively.

In the algorithm, a priori system knowledge is treated as assumptions/restrictions. An abstraction what we call "overlay level component interaction" is introduced. Different component behavior modes (namely reader, writer and transfer) are defined to explain the whole system's behavior by simple local component set behavior. Clearly, higher amount of information (i.e. state/topology change history) about the system results in better understanding/description of the identified system.

Finally, the existence of a GNA identification procedure for at least systems with defined assumptions and restrictions is shown and as to our knowledge, this is the first study, which is based on GNA identification. In fact, automated identifiability of systems (under the assumptions in section 3.1) of different scales ranging from nonliving systems (e.g. communication networks), cellular/species level biological/ecological systems to human systems through devising computer

procedures/algorithms is still questionable. This is because of known limitations of computing based on formal description of algorithms built over the Church-Turing thesis. The existence of alternative/unconventional hyper computing approaches and their products may result in contribution to the effort. However, it seems that the success is technically upper bounded by human-being himself/herself who is complicated enough to be understood.

GCRIS

REFERENCES

1- Sayama, H., Generative Network Automata: A Generalized Framework for Modeling Complex Dynamical Systems with Autonomously Varying Topologies, Proc. of the 2007 IEEE Symposium on Artificial Life (CI-Alife 2007), pp 214-221.

2- Weisstein, Eric W. "Dynamical System." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/DynamicalSystem.html, February 2010.

3- Galor, O.: "Discrete Dynamical Systems", Brown University, 2005, pp. 3-4.

4- Wolfram, S.: 'Universality and Complexity in Cellular Automata', Physica D, 10, 1984, pp. 1-35.

5- Garzon, M., Models of Massive Parallelism Analysis of Cellular Automata and Neural Networks, Texts in Theoretical Computer Science, Springer-Verlag, Berlin Heidelberg, 1995.

6- Langton, C.: Studying Artificial Life with Cellular Automata, Physica D 22, 1986, pp. 120-149.

7- Tarantola, Albert. "Inverse Problem." From MathWorld, created by Eric W. Weisstein. http://mathworld.wolfram.com/InverseProblem.html, March 2010.

8- Kılıç, H., A Methodology for Reverse Engineering Automata Networks, Ph.D. Dissertation, Middle East Technical University, Department of Computer Engineering, 135 pages, Ankara, Turkey, 1998, http://www.atilim.edu.tr/~hurevren/PHDTHESIS_HUR.rar

9- Goles, E. and Martinez, S., Neural and Automata Networks: Dynamical Behavior and Applications, Kluwer Academic Publishers, The Netherlands, 1990.

10- Von Neumann, J., Theory of Self-Reproducing Automata, edited by A.W. Burks, University of Illinois, Urbana, 1966.

11- Kauffman, S.A. (1991) Antichaos and Adaptation, Scientific American, August 1991, pp. 78-84.

12- Özdemir, B and Kılıç H. (2009). A Local Behavior Identification Algorithm for Generative Network Automata Configurations, Proceedings of the 10th European Conference on Artificial Life (ECAL 2009), vol. LNCS/LNAI 5777, 5778 (to be published).

13- Conway J. H., Conway's Game of Life: Scientific American, October 1970, pp. 120-123.

14- Sipper M., Non-Uniform Cellular Automata: Evolution in Rule Space and Formation of Complex Structures (1994).

15- Alarcon T., Byrneb H.M, Mainia P.K, A cellular automaton model for tumour growth in inhomogeneous environment, Journal of Theoretical Biology 2003, pp. 257–274.

16- Biondini F., Bontempi F., Frangopol D.M., Malerba P.G.,Cellular Automata Approach to Durability Analysis of Concrete Structures in Aggressive Environments, Journal of Structural Engineering 2004, pp. 1724-1737.

17- Sipper M., Tomassini M., Computation in artificially evolved, non-uniform cellular automata, 1999.

18- Sinvhal S., Mapping textures on 3d terrains: a hybrid cellular automata approach. MSc. Dissertation, Texas A&M University, 87 pages, Texas, U.S.A, 2005, http://research.cs.tamu.edu/keyser/Papers/SinvhalThesis.pdf

19- Tomassini M., Perrenoud M., Cryptography with cellular automata, Applied Soft Computing Journal, 2001, pp. 151-160.

20- Langton, C. G., Self-reproduction in cellular automata, Physica D 1984, 10 pp. 135-144.

21- Ljung L., System Identification - Theory for the User, Prentice-Hall, Englewood Cliffs, N J, 1987, ISBN 0-13-881640, 519 pages.

22- Aster R., Borchers B., Thurber C., Parameter Estimation and Inverse Problems, Academic Press, 2005, ISBN 0-13-978-0120656042, 320 pages.

23- Tarantola A., Inverse Problem Theory and Methods for Model Parameter Estimation, Society for Industrial and Applied Mathematics, 2004, ISBN 0-13-978-0898715729, 352 pages.

APPENDIX A

PSEUDOCODE FOR ID_GNA ALGORITHM

```
1 For each component i
2 {
3 Set rule set T_i to ∅;
4 Set component i's overlay level neighbor set G_i to { i };
5 }
6 For each component i
7 {
8  If (mode = reader or mode = transfer)
9    Construct the sets P_t(i) by using L_t(j) of other components
10   in given sequence;
11 }
12 For each component i
13 {
14  t = 0;
15  While t < k do
16  {
17   Switch (mode)
18   {
19    case reader : Construct next rule u by
20     using P_t(i), P_{t+1}(i), C_t(j) and current G_i;
21     case writer  : Construct next rule u by
22     using L_t(i), L_{t+1}(i), C_t(j) and current G_i;
23     case transfer : Construct next rule u by
24     using P_t(i), L_{t+1}(i), C_t(j) and current G_i;
25   }
26   If (u is consistent with all the rules in T_i)
27   {
28    If (u ∉ T_i) Set T_i = T_i ∪ {u};
29    t = t + 1;
```

```
30    }
31    Else
32    {
33     Set rule set T_i to ∅;
34     Find next overlay level nearest neighbor
35     component n by using ordered set H;
36     Set G_i = G_i ∪ {n};
37     t = 0;
38     Start over while loop;
39    }
40   }
41 }
42 Return all sets T_i and graph G;
```

GCRIS