POLYTECHNIC UNIVERSITY
Department of Computer and Information Science

COMPUTER SIMULATION OF COMPLEX SYSTEMS USING AUTOMATA NETWORKS

K. Ming Leung

Abstract: Computer simulation of the dynamics of complex systems using automata networks is discussed.

Directory

Table of Contents

1. Introduction

A complex system is a system composed of a large number of different highly connected interacting elements. The majority of natural and artificial systems are of a complex nature. Some examples are:

1. The human brain is composed of approximately 10 billion cells, called neurons. These cells interact by means of electro-chemical signals through their synapses.
2. Computer systems use large number of different electronic components at the level of transistors and logic gates.
3. Many multi-phase and composite materials
4. Social, economical and financial systems.

The interest is in the time evolution (dynamics) of such systems which often show highly non-trivial and complex behaviors. In science, we believe that the underlying laws describing these complex systems must be rather simple. One of the greatest challenges is to understand how these complex behaviors can result from simple laws. Important insights can be obtained by studying automata networks.

Toc ◀◀ ▶▶ ◀ ▶ Back ◀ Doc Doc ▶

The approach of automata networks is to represent the system by a network of interacting elements. Each element is referred to as an automaton. All relevant quantities are modeled by discrete variables. Each automaton in the network can only take on discrete state values. These values change according to certain rules as time progresses. These rules are usually very simple and yet the dynamics of the network can be extremely intriguing. Time t is discretized so that it takes on only integer values, $t = 0, 1, \dots, n, \dots$. The idea of a discrete time of course is very natural in a computer, where the time is controlled by an internal clock which coordinates the changes in the states of the logic units.

The most crucial part of modeling is to identify the relevant variables to simulate, how these variables should be described, how they interact with each other and affect the dynamics of each variables and therefore determine the dynamics of the entire system. Approximations are often needed to reduce the problem into a form that can be described by a network of automata. However once the problem has the desire discrete form, the dynamics can be followed by computer simulation exactly since all operations involve exact integer or boolean values. So automata networks are examples of discrete dynamical systems that can be simulated exactly on a digital computer. In addition since all variables and parameters are discrete, computations can be carried out much faster.

In this chapter we will consider some basic definitions and properties of general automata and automata networks. A special class of such discrete models known as cellular automata, will be considered in more detail in the next chapter. They were originally introduced by von Neumann and Ulam in 1948 as an idealistic model of biological self-reproduction.

2. Formal Definition of an Automaton

An automaton is defined as an object having 3 discrete sets:

1. I, the set of inputs, i.
2. S, the set of internal states, s, and
3. O, the set of outputs, o,

and 2 mappings:

1. $S(i, s)$, the state change function, which maps the inputs and the states at time t to a new state one time step later, i.e. $s_{t+1} = S(i_t, s_t)$.
2. $O(i, s)$, the output function, which maps the inputs and the state at time t to the outputs one time step later.

In general there are many inputs, internal states and outputs, so i, s and o are vectors (integer arrays).

3. An Example of an Automaton

The following pulse counter which counts the number of pulses in an input signal within a time interval, is an example of an automaton with 2 inputs and 4 outputs. The input vector, i, has 2 components, i_1 and i_2, given by a reset and a signal. Each component is represented by a sequence of 0 and 1.

When the reset is 1, the state at the next time step is reset to zero. When the reset is 0, the state at the next time step is given by the sum of the present state and the present signal. Consequently the internal state of the automaton is given by the total number of pulses in the signal since the last reset was performed.

The output at time t is given by the binary representation of the state at time t-1. It is therefore the binary representation of the number of pulses received since the last time the internal state was reset. The output consists of 4 components each representing a bit of the binary number. The first component represents the most significant bit and the last component, the least significant bit.

The state mapping function can be written in the form

$$s_{t+1} = (1 - i_{1,t}) * (s_t + i_{2,t}). \tag{1}$$

For example, if the inputs are given by:

$$i_1(0), i_1(1), \ldots, i_1(11) = 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \ldots \tag{2}$$

$$i_2(0), i_2(2), \ldots, i_2(11) = 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, \ldots \tag{3}$$

The initial value of the internal state does not really matter, it can be either 0 or 1, since it is reset to 0 after just one time step.

time	0	1	2	3	4	5	6	7	8	9	10	11	12	13
reset	1	0	0	0	0	0	0	0	0	0	0	0	?	?
signal	0	0	1	0	1	0	1	1	0	1	0	0	?	?
state	*	0	0	1	1	2	2	3	4	4	5	5	5	?
MSB	*	*	0	0	0	0	0	0	0	0	0	0	0	0
2nd bit	*	*	0	0	0	0	0	0	0	1	1	1	1	1
3rd bit	*	*	0	0	0	0	1	1	1	0	0	0	0	0
LSB	*	*	0	0	1	1	0	0	1	0	0	1	1	1

4. More Specialized Automata

We will mainly be interested on automata for which the internal state and the output are the same. A simplified automaton is then defined by its sets of inputs and internal states and a transition function f which gives the output at time $t+1$ as a function of the inputs and internal state at time t. We can write the relationship as:

$$0_{t+1} = f(s_t), \tag{4}$$

in vector notation.

Moreover, we will restrict ourselves to binary automata, where the inputs and internal states have only 2 possible values, 0 or 1.

There are 2 types of automata classified according to the type of transition functions used.

1. Boolean automata have transition functions given by boolean logic functions.
2. Threshold automata use the Heaviside unit function containing weights and threshold as the transition function. This type of automata is used mostly for neural networks.

5. Boolean Automata

Boolean automata have transition functions specified by boolean functions that operate on boolean variables and output boolean values. Boolean variables can take on only 2 possible values: true or false. Internally the value of true is represented by 1 and the value of false is represented by 0.

The number of inputs, k, specifies the connectivity of the automata. For a given k, the total number of possible input patterns is 2^k. Each of these input has 2 possible output values. Therefore the total number of different truth table is 2^{2^k}, each specifying a different automaton. Examples of boolean transition functions with 2 boolean input variables are the logic functions AND, OR and XOR. These transition functions are usually defined by a truth table giving the output for every input combinations. For example the truth table for the AND logic function is

input	11	10	01	00
output	1	0	0	0

For example if an automaton has connectivity of $k = 2$, then there are a total of $2^{2^2} = 16$ different boolean automata. The transition function can be conveniently described by the decimal representation of the output bits. The convention is that the leftmost value corresponds to the automaton with the highest number, just as in decimal notation where the leftmost digit represents the highest power of 10.

The sixteen boolean automata with 2 inputs are defined by the following 16 truth tables:

input	11	10	01	00	11	10	01	00
output	0	0	0	0	0	0	0	1
decimal	0				1			
name	Contradiction				NOR			

input	11	10	01	00	11	10	01	00
output	0	0	1	0	0	0	1	1
decimal	2				3			
name					Inverse 2			

input	11	10	01	00	11	10	01	00
output	0	1	0	0	0	1	0	1
decimal	4				5			
name					Inverse 1			

input	11	10	01	00	11	10	01	00
output	0	1	1	0	0	1	1	1
decimal	6				7			
name	XOR				NAND			

input	11	10	01	00	11	10	01	00
output	1	0	0	0	1	0	0	1
decimal	8				9			
name	AND				EQUvalence			

input	11	10	01	00	11	10	01	00
output	1	0	1	0	1	0	1	1
decimal	10				11			
name	Transfer 1				2 implies 1			

input	11	10	01	00	11	10	01	00
output	1	1	0	0	1	1	0	1
decimal	12				13			
name	Transfer 2				1 implies 2			

input	11	10	01	00	11	10	01	00
output	1	1	1	0	1	1	1	1
decimal	14				15			
name	OR				Tautology			

The names of the automata are taken from mathematical logic. The two functions numbered 0 and 15 have outputs which are independent of their inputs. Four functions, 3, 5, 10, and 12 depend only on one of the two inputs, which they either transmit or invert. The functions with decimal codes 1, 4, 7, 8, 11, and 13 are referred to as forcing functions, because when at least one of their outputs is in a certain state, the output does not depend on the state of the other input. For example, it suffices that one of the inputs of the AND function be in the state 0 for the output to be 0. Finally, for the two remaining functions, 6 and 9, the output can only be determined if both of the inputs are known.

6. Connectivity Structures

There are 3 common connectivity structures:

1. Random Connectivity - Connectivity is randomly chosen.
2. Complete Connectivity - Every automaton is connected to every other automaton.
3. Cellular Connectivity - Automaton are distributed on a periodic array in 1, 2 or 3 dimensions, and the connections are between nearest neighbors.

7. Iteration Modes

Iteration mode refers to the precise sequence in which the states of the automata are chosen for update during a given time step.

In a sequential iteration, automaton changes its states one after the other in a pre-determined fashion. In the following example, the automaton changes it state starting with automaton 1, automaton 2, etc. Mathematically this iteration mode amounts to applying the following transition functions one following the other (in the order as shown) within one time step:

$$\begin{aligned}
s_1(t+1) &= f_1(s_1(t), s_2(t), \ldots, s_{N-1}(t), s_N(t)) \qquad (5)\\
s_2(t+1) &= f_2(s_1(t+1), s_2(t), \ldots, s_{N-1}(t), s_N(t))\\
s_3(t+1) &= f_3(s_1(t+1), s_2(t+1), \ldots, s_{N-1}(t), s_N(t))\\
&\cdots\\
s_N(t+1) &= f_N(s_1(t+1), s_2(t+1), \ldots, s_{N-1}(t+1), s_N(t))
\end{aligned}$$

In a synchronous iteration, the states of the automata are updated in parallel, based on their values in the previous time step. Mathematically this iteration mode amounts to applying the following transition functions in any order within one time step:

$$\begin{aligned} s_1(t+1) &= f_1(s_1(t), s_2(t), \ldots, s_{N-1}(t), s_N(t)) \qquad (6) \\ s_2(t+1) &= f_2(s_1(t), s_2(t), \ldots, s_{N-1}(t), s_N(t)) \\ s_3(t+1) &= f_3(s_1(t), s_2(t), \ldots, s_{N-1}(t), s_N(t)) \\ &\ldots \\ s_N(t+1) &= f_N(s_1(t), s_2(t), \ldots, s_{N-1}(t), s_N(t)) \end{aligned}$$

8. Automata Networks

An automata network is composed of a set of automata interconnected in such a way that the output of some are the inputs of others. It is therefore a directed graph where the nodes are the automata and the edges are the connections from the output of one automaton to the input of another.

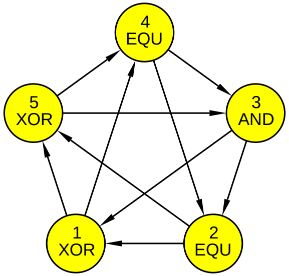

Figure 1: *A closed network consisting 5 boolean automata. Each automaton has 2 inputs and performs a logical function as shown.*

We consider here a closed automata network, that is it has no external connections. An example of such a network is shown in the figure. It is composed of 5 boolean automata each having 2 inputs. This network is equivalent to the following set of 5 logical relations:

$$\begin{aligned} s_1(t+1) &= XOR(s_2(t), s_3(t)) \\ s_2(t+1) &= EQU(s_3(t), s_4(t)) \\ s_3(t+1) &= AND(s_4(t), s_5(t)) \\ s_4(t+1) &= EQU(s_5(t), s_1(t)) \\ s_5(t+1) &= XOR(s_1(t), s_2(t)) \end{aligned} \tag{7}$$

if we adopt a parallel iteration mode. Suppose the initial configuration of the automata network is given by 00000 (again the leftmost bit corresponds to the automaton with the highest number). The states after one time step are given by

$$\begin{aligned} s_1(t+1) &= XOR(0,0) = 0 \\ s_2(t+1) &= EQU(0,0) = 1 \\ s_3(t+1) &= AND(0,0) = 0 \\ s_4(t+1) &= EQU(0,0) = 1 \\ s_5(t+1) &= XOR(0,0) = 0 \end{aligned} \tag{8}$$

So the network has configuration 01010. Notice that implementing the parallel iteration mode on a serial computer requires the use of 2 sets of state variables, one set to store the old values and another for the new values. At each time step, the old values are used to update the new values, and then the old values are replaced by the corresponding new ones.

On the other hand if we adopt a sequential iteration mode instead, then the states after one time step are given by

$$\begin{aligned} s_1(t+1) &= XOR(0,0) = 0 \\ s_2(t+1) &= EQU(0,0) = 1 \\ s_3(t+1) &= AND(0,0) = 0 \\ s_4(t+1) &= EQU(0,0) = 1 \\ s_5(t+1) &= XOR(0,1) = 1 \end{aligned} \tag{9}$$

Therefore the network will end up having a different configuration: 11010 one time step later. Implementation of the sequential iteration mode is straightforward and does not require an extra set of variables.

9. Successor Table

For a simple automata network, it is not too difficult to obtain a table showing how each possible configuration should be updated in a time step. Such a table is referred to as the successor table.

For our automata network consisting of 5 boolean automata, the total number of possible configurations is $2^5 = 32$. So the successor table contains 32 transition rules. Each configuration is labeled by its decimal code. The following successor table is obtained assuming the parallel iteration mode.

code	before	code	after	code	before	code	after
0	00000	10	01010	16	10000	2	00010
1	00001	18	10010	17	10001	26	11010
2	00010	27	11011	18	10010	19	10011
3	00011	3	00011	19	10011	11	01011
4	00100	9	01001	20	10100	1	00001
5	00101	17	10001	21	10101	25	11001
6	00110	24	11000	22	10110	16	10000
7	00111	0	00000	23	10111	8	01000
8	01000	8	01000	24	11000	4	00100
9	01001	16	10000	25	11001	28	11100
10	01010	25	11001	26	11010	21	10101
11	01011	1	00001	27	11011	13	01101
12	01100	11	01011	28	11100	7	00111
13	01101	19	10011	29	11101	31	11111
14	01110	26	11010	30	11110	22	10110
15	01111	2	00010	31	11111	14	01110

10. Iteration Graphs

Since a boolean automaton has 2 possible states, a network of N boolean automata has a total of 2^N possible configurations. Each configuration is specified by a unique combination of the state of each automaton in the network. Starting with a given initial configuration, the time evolution (dynamics) of the network can be followed by applying the transition rules given by the successor table to change the states of the automata at each time step.

We can then draw a directed graph, called the iterated graph, where the nodes label the configurations of the automata network in decimal codes, and the directed edges indicate the transition of the network from its configuration at a given time to the next time.

We can start with any initial configuration and follow its time evolution. We have to be sure that all possible configurations are visited. Because of the fully deterministic nature of the transition rules, we stop the iteration as soon as we come to a configuration that has been previously visited. We then choose an initial configuration that has not yet been visited and iterate from there. The process is terminated once all possible configurations have been visited. For a simple automata network, such as the one we are considering here, the entire dynamics of the network can be found this way.

For our boolean automata network for example, we can start with configuration 0. We then go to configurations 10, 25, 28, 7 and back to 0 again. We can stop the iteration because we are clearly going to go around in circles among these 5 configurations if we continue any further. This kind of cyclic dynamic behavior is called a limit cycle.

Let us start all over again but this time with a different initial configuration, say 1. From there we go to configurations 18, 19, 11 and back to 1. We end up in another limit cycle. But this time the limit cycle involves only 4 different configurations.

We repeat again with a new configuration, say 2. This configuration goes into configurations 27, 13, and 19. We can stop the iteration since configuration 19 has been visited previously. The results obtained in the above two steps can be combined to obtain the following iterated graph:

$$\begin{tikzcd}
2 \arrow[r] & 27 \arrow[r] & 13 \arrow[r] & 19 \arrow[d] & 18 \arrow[l] \\
 & & & 11 \arrow[r] & 1 \arrow[u]
\end{tikzcd} \tag{10}$$

Notice that two separate branches merge into configuration 19, one from configuration 13, and the other from 18. However the deterministic nature of the transition rules precludes the possibility of branching out from any configuration.

It is clear that we are interested in how one configuration evolves into another, and so the precise positioning of the configurations in a graph is totally irrelevant. The topology of the connection pathways is all that matters here.

Next we start with configuration 3, which we have not encountered before. Note that according to the transition rules, configuration 3 goes into itself. Configuration 3 is referred to as a fixed point in the dynamics of the network, since it does not change once we get to this configuration. An inspection of the successor table reveals that no other configuration evolves into configuration 3. Consequently this fixed point is called an isolated fixed point. Configuration 8 is another fixed point of the dynamics of this network, however it is not an isolated fixed point since configuration 23 evolves into it.

After all the configurations have been visited, we see that the iterated graph consists of 4 separate sub-graphs, breaking up the configurations into 4 distinct groups. Each sub-graph represents a basin of attraction because configurations within a sub-graph can only evolve into other configurations within the same sub-graph, and can never evolve into a configuration belonging to another sub-graph.

If the evolution of each configuration within a basin is followed, it always ends up in some kind of infinite loop, no matter which configuration within the basin is chosen as the initial configuration. These ultimate asymptotic dynamical behaviors are referred to as attractors of the dynamic system. The lengths of these infinite loops are called the periods of the attractors. A period-1 attractor is called a limit point. Attractors having periods larger than 1 are called limit cycles. The totality of configurations which converges toward an attractor constitute a basin of attraction. There is one and only one attractor

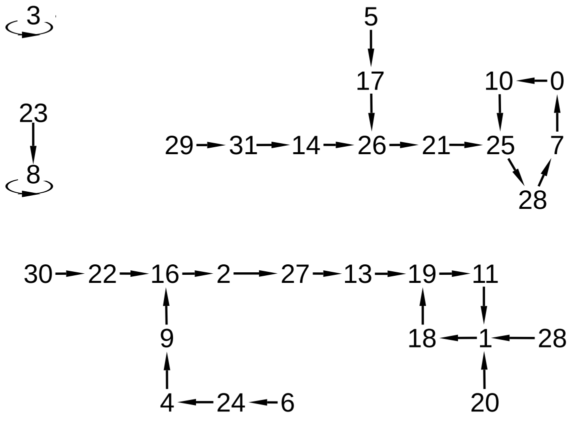

Figure 2: *The iterated graph showing the dynamics of a closed automata network consisting of 5 boolean automata.*

for each basin of attraction.

In this example, there are 4 basins of attraction, two have limit points and the other two have limit cycles, of periods 4 and 5.

Clearly it is possible to construct a complete iteration graph only for simple automata networks. For large ($N \gg 1$) networks with complicated connectivities ($k \gg 1$), we must be content only with describing the dynamics of the network by characterizing the basins of attraction and the associated attractor.

Therefore the general goals are to determine:

1. the number of different attractors
2. the periods of each attractor
3. the number of configurations within the basin of each attractor
4. the average duration of the transients, that is the average time of evolution from the initial configuration (Garden of Eden) to the attractor (Heaven or Hell).

Of course computer simulation is an important tool for such studies.

References

[1] G. Weisbuch, tr. by S. Ryckebysch, *Complex Systems Dynamics: An Introuction to Automata Networks*, Addison-Wesley publishing Company, 1990.

[2] S. Wolfram, Theory and Applications of Cellular Automata, World Scientific, 1986.

[3] H. Gould and J. Tobochnik, *An Introduction to Computer Simulation Methods: Applications to Physical Systems*, Second Edition, Ch. 11, Sec. 6. (Addison Wesley, New York, 1996).