Session 11

Turing's Lure, Gödel's Curse

Abouda Abdulla
Helsinki University of Technology
Communication Engineering Laboratory

abouda@cc.hut.fi

Linear systems are well understood and completely predictable, whatever the initial state and no matter what is the system dimension. But if the system is nonlinear then different situation will be faced, and the theory of linear systems no longer is valid. It has been shown that the resulting behaviors from simple rules might emerge to form very complicated structure [6][10]. For example, the emerging behaviors from non-linear continuous time system or the emerging behaviors from hybrid systems or even from cellular automata can be theoretically intractrable. The complexity here means that these systems will forever defy analysis attempts [10].

In this article we will be elaborating on these issues by considering cellular automata as an example.

11.1 Computability theory

Computability theory dates back to 1930's when mathematicians began to think about what it means to be able to compute a function. That was before the advent of digital computers. One of those mathematicians is Alan Turing (1912–1954) who is the inventor of Turing machines [1]. The conclusion that

Alan Turing came out with was:

> *A function is computable if it can be computed by a Turing machine.*

11.2 Gödelian undecidability

"All frame works that are powerful enough are either incomplete or inconsistent". This is what is known as Gödelian undecidability principle. This principle has two interpretations, in mathematics and in physics. In mathematics it says that there does not exist any reasonable finite formal system from which all mathematical truth is derivable. And in physics it says that with respect to predication, there are some systems that cannot be completely predicted — like computation in Turing machine or predictbility of Life pattern in the game of Life. In this article we consider the physical interpretation [2].

It has been shown that any formalism that is powerful enough suffers from this Gödelian undecidability problem. A prototype of such a powerful system is *Turing machine.*

11.3 Turing Machine

A Turing machine is a very simple machine that has all the power that any digital computer has. A Turing machine is a particularly simple kind of computer, one whose operations are limited to reading and writing symbols on a tape, or moving along the tape to the left or right. This tape is divided into squares, any square of which may contain a symbol from a finite alphabet, with the restriction that there can be only finitely many non-blank squares on the tape. At any time, the Turing machine has a read/write head positioned at some square on the tape. Figure 11.1 shows how the tape looks like with a series of A's and B's written on the tape and with read/write head located on the rightmost of these symbols.

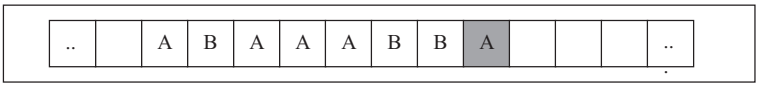

Figure 11.1: A tape with series of A's and B's

Furthermore, at any time, the Turing machine is in any one of a finite number of internal states. The Turing machine is specified by a set of instructions of the following form:

(current state, current symbol, new state, new symbol, left/right)

The instruction means that if the Turing machine is now in current state, and the symbol under the read/write head is current symbol, then the Turing machine changes its internal state to a new state, replaces the symbol on the tape at its current position by a new symbol, and moves the read/write head one square in the given direction left or right. It may happen that the Turing machine is in a situation for which it has no instruction and that makes the Turing machine to halt. Defining instructions of Turing machine can be though of as programming the Turing machine.

There are several conventions commonly used. The convention that numbers are represented in unary notation is adopted here. It means that the non-negative integer number n is represented by using of successive 1's. Furthermore, if a function $f(n_1,n_2,...,n_k)$ to be computed, we assume that initially the tape consists of $n_1,n_2,...,n_k$, with each sequence of 1's separated from the previous one by a single blank, and with tape head initially located at the right most bit of the first input argument, and the state of the Turing machine in some initial specified value. We say that the Turing machine has computed $m = f(n_1,n_2,...,n_k)$ if, when the machine halts, the tape consists of the final result and the read/write head at the right most bit of the result.

Example 1: Multiplication using Turing machine

Suppose we want to create a Turing machine to compute the function $m = multiply(n_1,n_2)=n_1 \times n_2$. If $n_1= 3$ and $n_2= 4$, then the tape in the beginning of computation will look like Fig. 11.2.

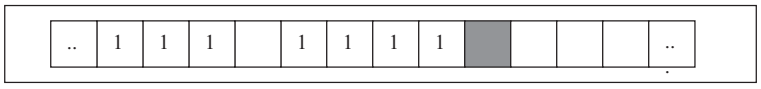

Figure 11.2: The tape in the beginning of computation

Here the position of read/write head is marked with dark cell. After the computation is finished the Turing machine should halt with its tape looking as Fig. 11.3, giving the final result ($m = 3 \times 4 = 12$).

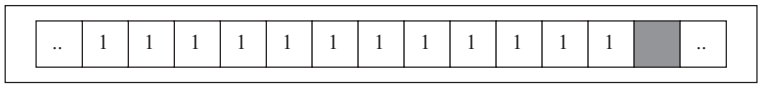

Figure 11.3: The tape at the end of computation

Due to the simplicity of Turing machine, programming one machine to perform a specific computation task is challenge [1][2][3][4] [1].

11.4 Computation as frame work

Cellular automata and other systems with simple underlying rules can produce complicated behaviors. To analyze the behavior of these systems, our experience from traditional science might suggest that standard mathematical analysis should provide the appropriate basis for any framework. This kind of analysis tends to be useful only when the overall behavior of the system is fairly simple. So what can one do when the over all behavior is more complex? The main purpose of Wolfram's book "A New Kind of Science" is to develop a new kind of science that allows progress to be made in such cases. The single most important idea that underlines this new science is the notation of computation. So far we have been thinking of cellular automata and other systems as simple computer programs. Now we will think of these systems in terms of the computations they can perform [5].

In a typical case, the initial conditions for a system like cellular automaton or Turing machine can be viewed as corresponding to the input to a computation, while the state of the system after some number of steps corresponding to the output. The key idea is to think in purely abstract terms about the computation that is performed. This abstract is useful due to two reasons, the first, because it allows us to discuss in a unified way systems that have completely different underlying rules, and the second reason, because it becomes possible to imagine formulating principles that apply to a very wide variety of different systems [5].

11.4.1 Computations in cellular automata

Cellular automaton can be viewed as a computation. The following examples show how cellular automata can be used to perform some computations [5].

[1] A Visual Turing Machine simulation program is available from the web page `http://www.cheran.ro/vturig`

Example 2: Computing whether a given number is even or odd

If one starts cellular automaton shown in Fig. 11.4 with an even number of black cells, then after a few steps of evolution, no black cells are left. But if instead one stars it with an odd number of black cells, then a single cell survives forever. Using this cellular automaton we can check whether the given number is even or odd.

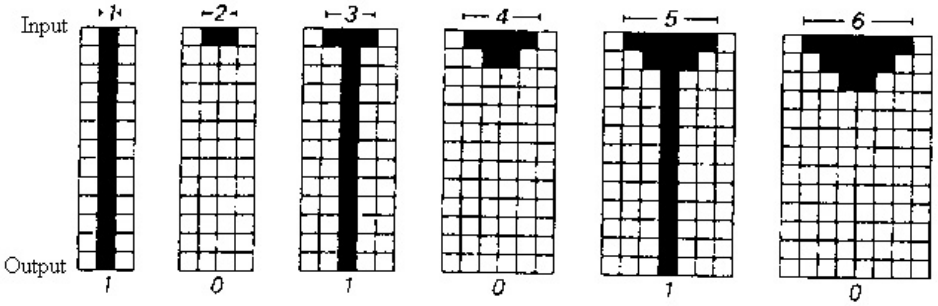

Figure 11.4: Cellular automaton to check whether the given number is odd or even

But cellular automaton can be used to perform more complicated computations, too.

Example 3: Computing the square of any number

The cellular automaton shown in Fig. 11.5 can be used to calculate the square of any given number. If one starts with 5 black squares, then after a certain number of steps the cellular automaton will produce a block of exactly $5 \times 5 = 25$ black squares.

Example 4: Computing the successive prime numbers

The rule for this cellular automaton (Fig. 11.6) is somewhat complicated; it involves a total of sixteen colors possible for each cell.

But the cellular automata that have been presented so far can be conveniently described in terms of traditional mathematical notations. But what kinds of computations are cellular automata like the one presented in Fig. 11.7 performing? Of course we cannot describe these computations by anything

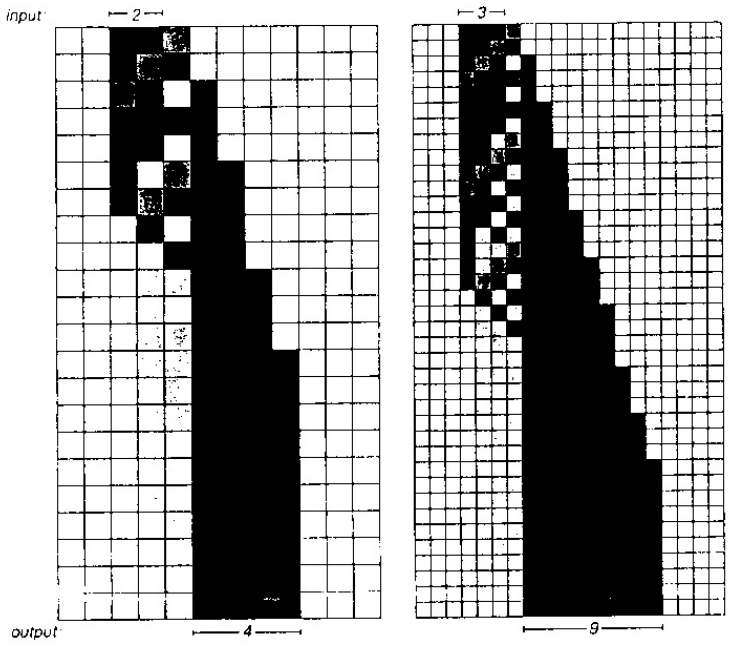

Figure 11.5: Cellular automaton that computes the square roots of any number

as simple as saying, for example, they generate primes. So how then can we ever expect to describe these computations? [5]

11.5 The phenomena of universality

Universal system is a system whose underlying construction remains fixed, but which can be made to perform different tasks just by being programmed in different ways. For example, take the computer: The hardware of the computer remains fixed, but the computer can be programmed for different tasks by loading different pieces of software.

If a system is universal, then it must effectively be capable of emulating any other system, and as a result it must be able to produce behavior that is as complex as the behavior of any other system.

As we have seen before simple rules can produce complex behavior. We will see later how simple rules can produce universal system. With their simple and rather specific underlying structure one might think that cellular automata would never be capable of emulating a very wide range of other systems. But what we will see in this section is that cellular automata can be

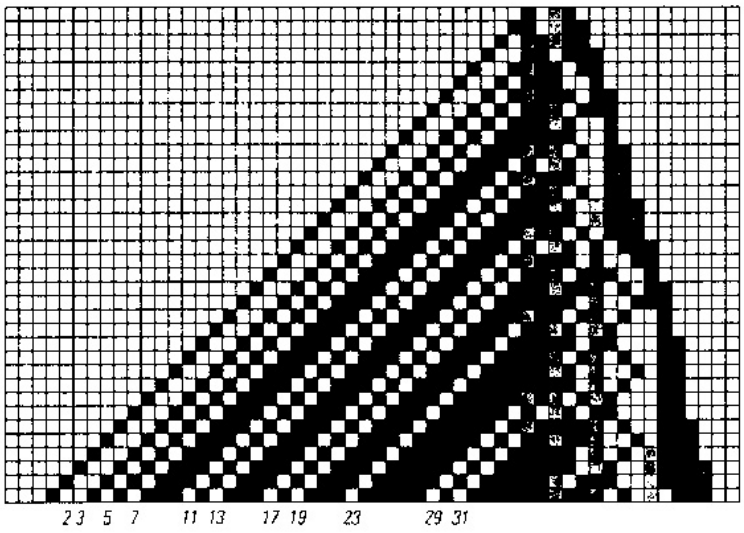

Figure 11.6: Cellular automaton to compute successive prime numbers

made to emulate computer. We will consider cellular automata with specific rules known as Game of Life [6][7][8][9].

Because the construction of a "Life Computer" is an involved task, a thorough discussion is needed here.

11.6 Game of Life

One form of cellular automata is Game of Life, which was invented by Cambridge mathematician John Conway. Game of Life is no-player game. Life is played on an infinite squared board. The cells in this game represent the population. At any time some of the cells will be live and others will be dead. The initial structure of live cells can be set as one wants but when time starts to go on the cells birth or death is controlled by rules [6]. The rules of this game are as follows[2]:

- Birth: A cell that is dead at time t becomes live at time $t + 1$ only if exactly three of its eight neighbors were live at time t.
- Death by overcrowding: A cell that is live at time t and has four or more of its eight neighbors live at time t will be dead by time $t + 1$.

[2]The Game of Life simulation program is available, for example, at the web page `http://www.bitstorm.org`

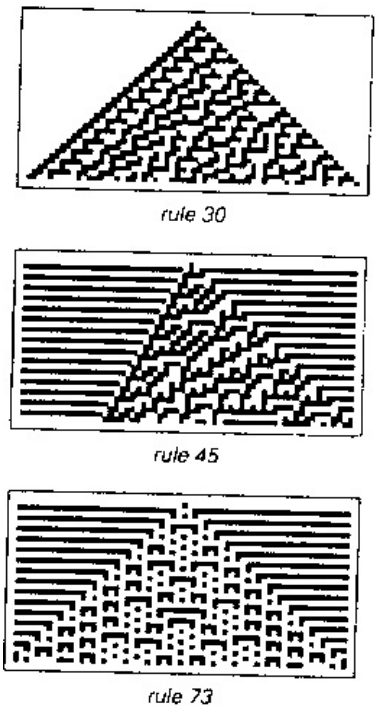

Figure 11.7: Cellular automata that can not be described mathematically

- Death by exposure: A live cell that has only one live neighbor, or none at all, at time t, will also be dead at time $t + 1$.
- Survival: A cell that was live at time t will remain live at time $t + 1$ if and only if it had just 2 or 3 live neighbors at time t.

An example of typical Life history is shown in Fig. 11.8. The starting generation has five live cells G0. In the first generation G1 there are three cells on either side of the line that are dead at G0, but have exactly three live neighbors, so will come to life at G1. From G1 to G2 the corner will survive, having 3 neighbors each, but every thing else will die by overcrowding. There will be 4 births, one by the middle of each side. From G2 to G3 there is a ring which each live cell has 2 neighbors so everything survives, there are 4 births inside. From G3 to G4 overcrowding will kill all cells except the 4 outer cells and the neighbors of these cells will born. From G4 to G5 another survival ring with 8 happy events about to take place. From G5 to G6 more overcrowding again leaves just 4 survivors. This time the neighboring births form: From G6 to G7 four separated lines of 3, called **Blinkers**, which will never interact again. From G7 to G8, G9, G10 - at each generation the

tips of Blinkers die of exposure but the births on each side reform the line in a perpendicular direction. The configuration will therefore oscillate with period two forever. The final pair of configuration known as Traffic Lights.

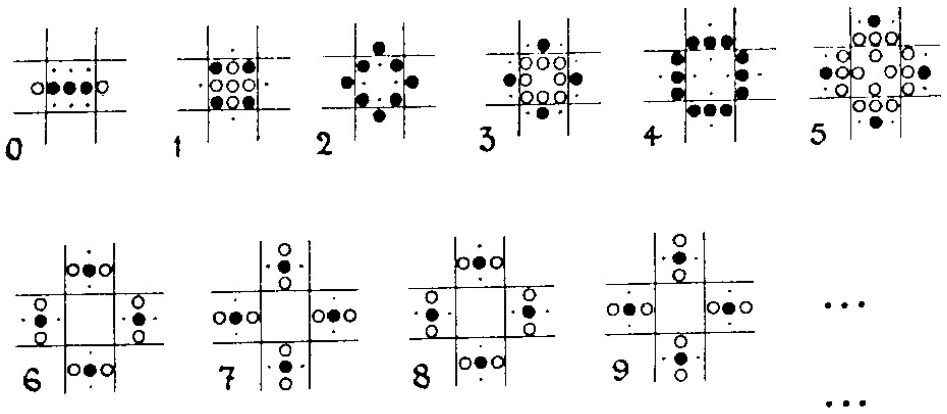

Figure 11.8: Typical Life history

Still Life

There are some common forms of still lives; some of these are shown in Fig. 11.9, with their traditional names. The simple cases are usually loops in which each live cell has two or three neighbors according to local curvature, but the exact shape of the loop is important for effective birth control.

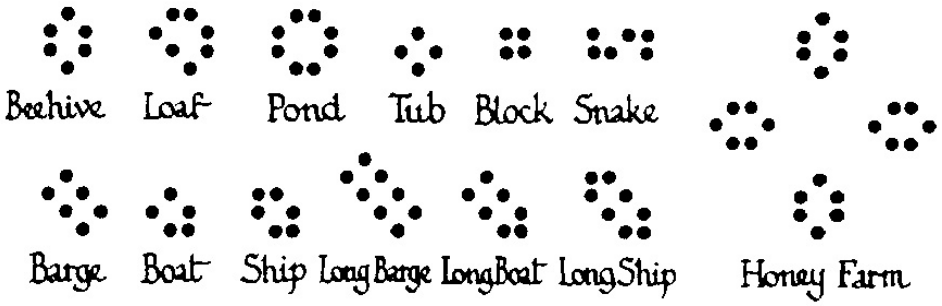

Figure 11.9: Some of the common forms of Still Life

Life Cycle

There are some configurations whose Life history repeats itself after some time. This period of time known as Life Cycle. The Blinker is the simplest example of these configurations, where the Blinker repeats itself with period larger than 1. Some configurations with their own Life cycles are shown in Figs. 11.10 and 11.11.

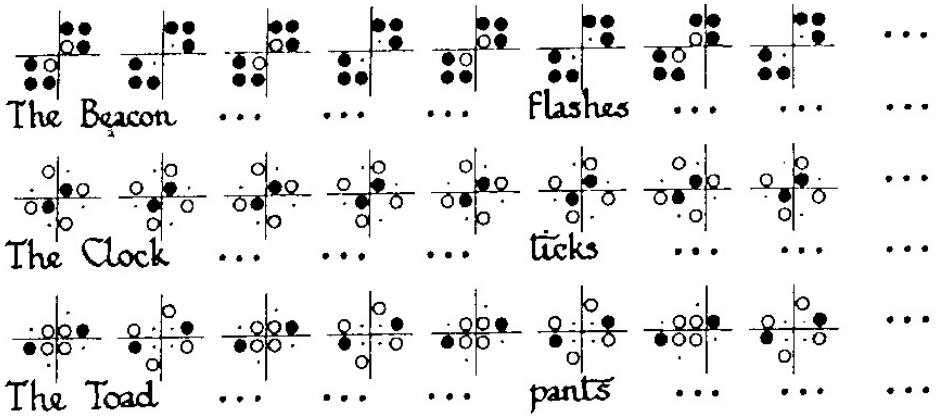

Figure 11.10: Three Life cycles with period two

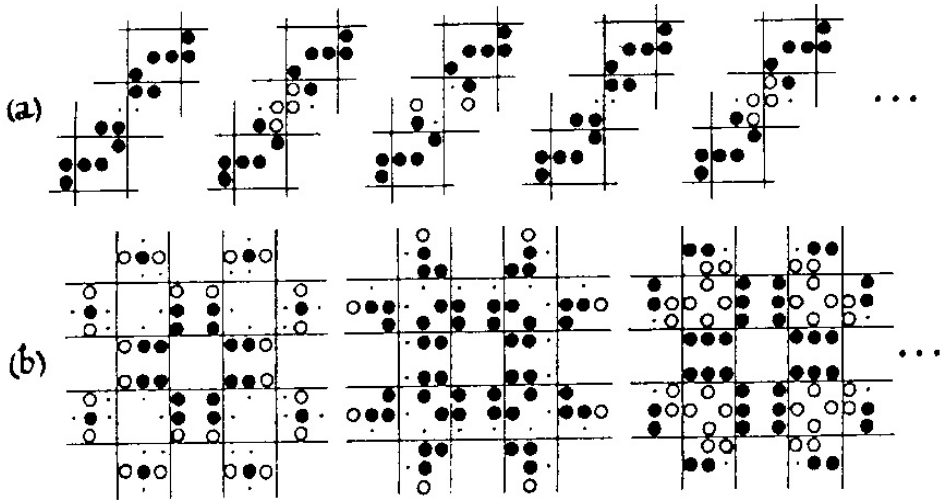

Figure 11.11: Life cycle with period three: (a) Two eaters Gnash at each other. (b) The Cambridge Pulsar

The Glider and other space ships

If we look at Fig. 11.12, we can notice that the generation 4 is just like generation 0 but moved one diagonal place, so that the configuration will steadily move across the plane. The arrangements at times $2, 6, 10, \ldots$ are related to those at times $0, 4, 8, 12, \ldots$ by symmetry that geometers call a glide reflexion — so this creature is known as "Glider".

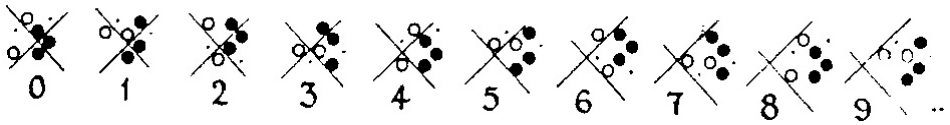

Figure 11.12: The Glider moves one square diagonally each four generations

The unpredictability of Life

Is it possible to tell before hand the destiny of a Life pattern? Is it going to fade away completely? Or is it going to be static? Or it will travel across the plane, or its going to expand indefinitely? To answer these questions let us have a look at very simple configuration, a straight line of n live cells.

> When $n = 1$ or 2 the Life pattern fades immediately. When $n = 3$ the result is the Blinker. When $n = 4$ it becomes a Beehive at time 2. When $n = 5$ it becomes Traffic Light at time 6. When $n = 6$ it fades at time 12. When $n = 7$ it makes a beautifully symmetric display before terminating in the Honey Farm at time $t = 14$. When $n = 8$ it gives 4 blocks and 4 Beehives. When $n = 9$ it makes two sets of Traffic Lights, and so on. Even when we start with very simple configuration and small number of cells it is not easy to see what goes on [6].

The other question of Life is that, can the population of a Life grow without limit? The answer of this question was YES and a group at M.I.T proved it in November 1970. The conclusion from this part is that, Life is really unpredictable.

11.6.1 Making a Life computer

Many computers have been programmed to play the game of Life. In this section we will see how to define Life patterns that can imitate computers. Computers are made from pieces of wire along pulses of electricity go. To mimic these by certain lines in the plane along which Glider travel (see Fig. 11.14.

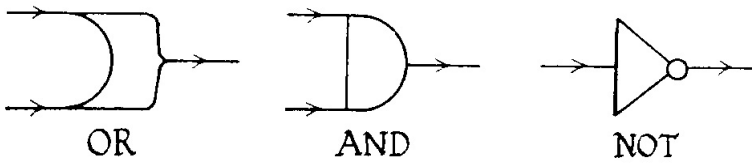

Figure 11.13: The three logical gates

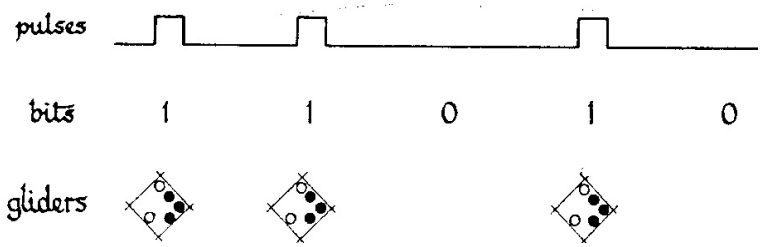

Figure 11.14: Gliding pulses

In the machine there is a part called clock, this clock generating pulses at regular intervals. The working parts of the machine are made up of logical gates. The Glider Guns can be considered as pulse generators. To understand how to construct logical gates we first study the possible interactions of two Gliders, which crash at right angles [6].

When Glider meets Glider

The figure below (Fig. 11.15) shows two Gliders crash in different ways:

1. To form a Blinker
2. To form a Block
3. To form a Pond
4. To annihilate themselves.

The last crashing way has special interest, because the vanishing reactions turn out to be surprisingly useful.

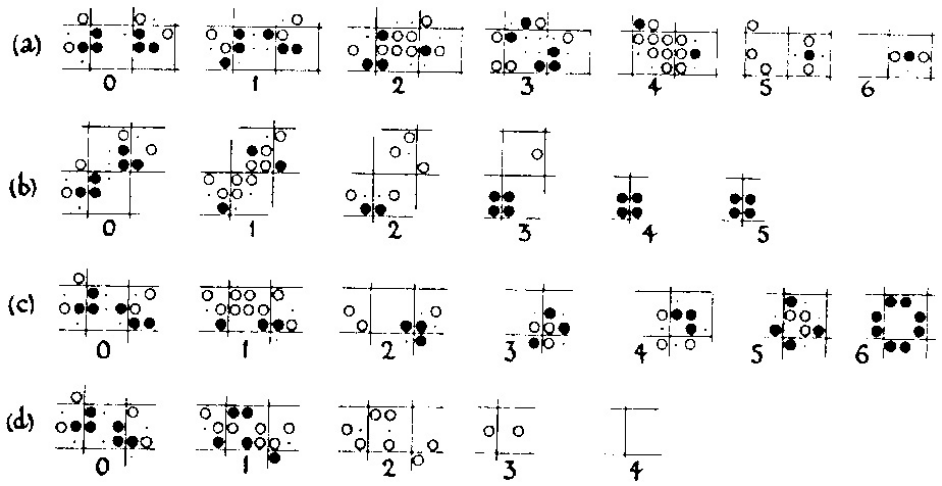

Figure 11.15: Gliders crashing in diverse fashion

How to make a not gate

The vanishing reaction can be used to make a not gate. As shown in Fig. 11.16, the input stream enters at the left of the figure and the Glider Gun is positioned and timed so that every space in the input stream allows just one Glider to escape from the gun [6][9].

The Eater

Other phenomena that happens when two Gliders meet, an Eater will be constructed. This is shown in Fig. 11.17.

Gliders can build their own guns

So far we have been concerned by Glider meets another Glider, what happens when a Glider meets other thing? More constructively it can turn a Pond into a Ship (Fig. 11.18) and a Ship into a part of the Glider Gun. And since Gliders can crash to make Block and Pond they can make a whole Glider Gun [6].

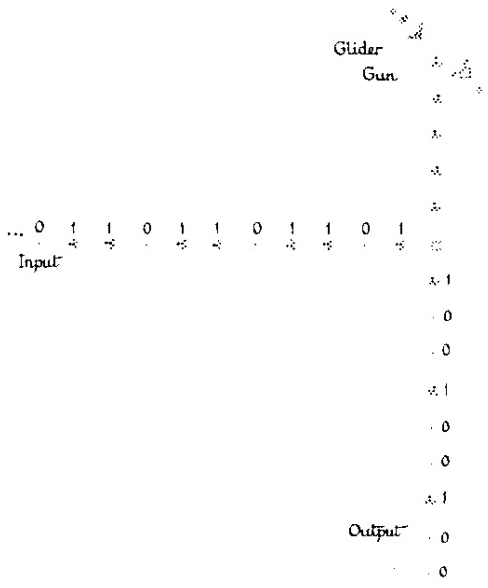

Figure 11.16: A Glider Gun and vanishing reaction make a NOT gate

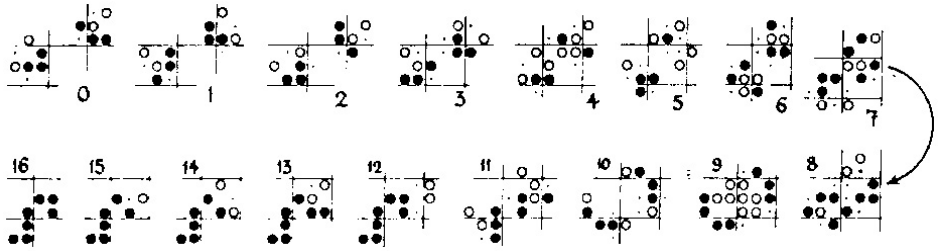

Figure 11.17: Two Gliders crash to form an Eater

The kickback reaction

Other useful reaction between Gliders is the kickback (Fig. 11.19) in which the decay product is a Glider travelling along a line closely parallel to one of the original ones but in the opposite direction. We may think of this Glider as having been kicked back by the other one [6][9].

Building blocks for our computer

Figure 11.20 shows logical gates constructed based on vanishing reactions. From here on it is just an engineering problem to construct an arbitrarily large finite computer. Such computers can be programmed to do any thing [6] (construction of memory elements, for example, has been skipped here).

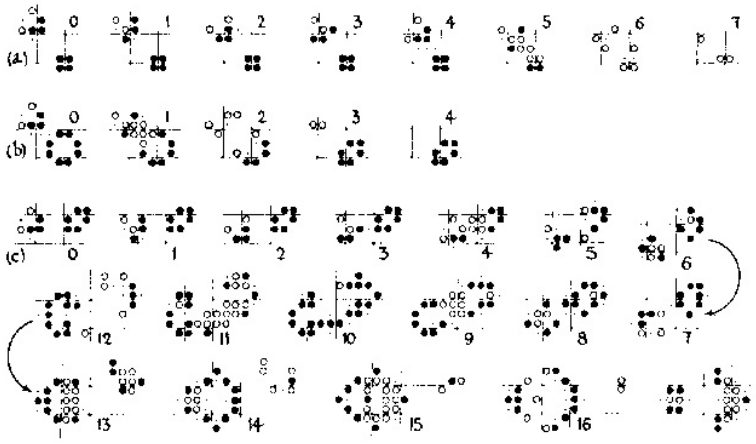

Figure 11.18: In (b), Glider dives into Pond and comes up with Ship. In (c), Glider crashes into Ship and makes part of Glider Gun

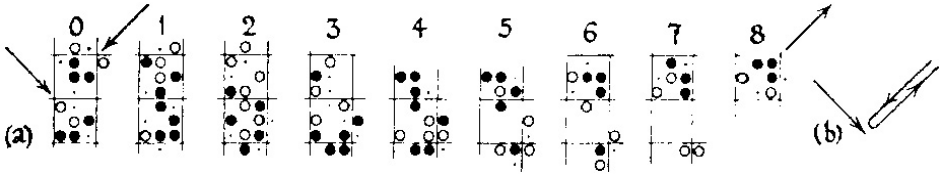

Figure 11.19: The kickback

11.7 Conclusion

The conclusion that can be drawn is that complex systems like cellular automata or Turing machine with very simple underlying rules are universal systems which can emulate other complicated systems. When we come to analyze such systems, we can see clearly that we do not have a framework to deal with these systems. New kind of science to analyze these powerful systems is needed.

Bibliography

1. http://www.cheran.ro/vturig

2. http://tph.tuwien.ac.at/ svozil/publ/1996-casti.htm

3. http://www.turing.org.uk/turing/

4. http://www.bitstorm.org

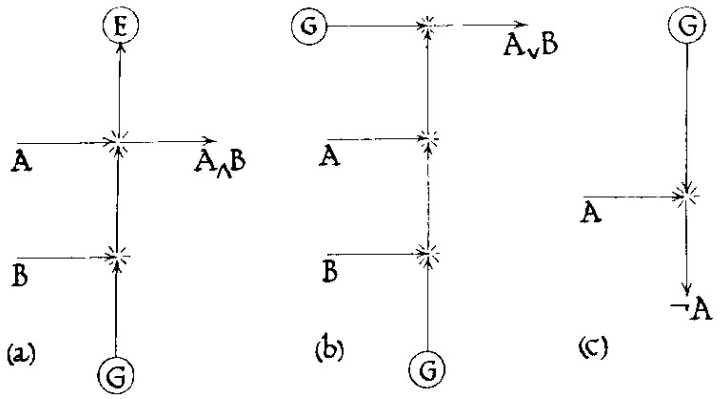

Figure 11.20: (a) AND gate, (b) OR gate, and (c) NOT gate

5. Wolfram, S.: A New kind of science.Wolfram Media, Champaign, Illinois, 2002.

6. Elwyn Berlekamp, Conway, J., and Guy, R.: "Winning Ways", Vol. 2, Chapter 25.

7. http://www.ams.org/new-in-math/cover/turing.html

8. http://www-csli.stanford.edu/hp/turing1.html

9. http://ddi.cs.uni-potsdam.de

10. Hyötyniemi, H.: On universality and undecidability in dynamic systems. Helsinki University of Technology, Report 133, December 2002.