Chapter 2
Background

Abstract The background material is presented giving a description of Conway's Game of Life, Turing machines and Counter Machines. The details of the rules for Game of Life are given along with the details of some of the most well known patterns. The operational cycle of a Turing machine is explained as well as an outline for a universal Turing machine. A simple Turing machine is presented while introducing the notation used to describe Turing machines in later chapters. Counter machines are described with an example instruction set and very simple program. Conway's proof of the universality of the Game of Life is outlined with descriptions of the logical components he proposed.

2.1 Conway's Game of Life

The 'Game of Life' (GoL) invented by John Conway is a cellular automaton. It was popularised through Martin Gardner's articles in Scientific American in the 1970s [1].

A cellular automaton is one of the simplest mathematical models to have properties of space and time. It is a machine made up of an ordered array of cells. The only thing that changes is a property of the cells called state. All the cells are identical in construction and can be in one state of a finite set. The cells change state according to a small set of transition rules which specify the next state of a cell according to the states of its neighbouring cells. These rules are applied to all the cells at the same time to make discrete time steps called generations. Some variants of Cellular Automata are non deterministic in that an initial pattern can lead to more than one outcome. GoL is a member of the more conventionally deterministic kind where every pattern has just one successor pattern.

Mathematically cellular automata are interesting because of the way simple patterns evolve. There are an infinite variety of Cellular automata made up of different arrangements of cells and different rules. The astonishing thing is the complex behaviour of these patterns with very simple spatial arrangements of cells and very simple rules.

P. Rendell, *Turing Machine Universality of the Game of Life*,
Emergence, Complexity and Computation 18, DOI 10.1007/978-3-319-19842-2_2

One of the first people to study cellular automata was von Neumann in the 1940s. He was interested in self replicating machines and designed machines with quite large numbers of states to enable self replicating machines to be built with small numbers of cells.

The simplest cellular automata have a one dimensional array of cells with just two states have been studied by Wolfram [2]. One of these known as rule 110 has been proved by Cook [3] to support universal computation. It has been used to build some of the smallest universal Turing machines as outlined in Sect. 3.8.

As part of his study of one dimensional cellular automata Wolfram [4] developed a scheme for classification of cellular automata into four classes. The fourth of these is described as a cellular automaton in which: 'Evolution leads to complex localized structures, sometimes long-lived'. This is the category in which GoL falls. GoL is in fact one of the most interesting cellular automata because of the rich patterns it supports. It seems to be poised mid way between the class of cellular automata in which most pattern quickly evolve to stable short period oscillators or nothing and the class of cellular automata in which most patterns expand to fill the universe.

In Conway's Game of Life each cell has just two possible states, live and dead. The spatial arrangement for the cells is an infinite two dimensional square grid pattern. The rules are:

- If a live cell has two or three live neighbouring cells, then it will remain alive in the next generation, otherwise it will die.
- If a dead cell has exactly three live neighbouring cells then it will come to life in the next generation.

A neighbour to a cell is one of the eight cells which touch it. Figure 2.1 shows the neighbourhood counts for the simple period two oscillating pattern known as a blinker. Figure 2.2 shows two still life patterns, these are patterns which do not change from one generation to another.

Figure 2.3 shows one of the most important patterns, the glider. This is one of several patterns which reproduce themselves with an offset in space, that is they appear to move. One very useful reaction of two gliders is the kickback reaction shown in Fig. 2.4 where one glider is reflected 180° by another glider. The glider is said to be kicked back.

Another moving pattern is the spaceship, there are three simple versions shown in Figs. 2.5, 2.6 and 2.7.

Figure 2.8 shows six of the fifteen generations of an oscillator called a pentadecathlon. It generates a small pattern which separates from the main oscillator. This is called a spark as it dies by itself but may interact with other objects without effecting the oscillation of the pentadecathlon.

Figure 2.9 shows the pattern which first made the idea of universality in the Game of Life seem possible. This is the Gosper glider gun found by Bill Gosper in 1970.

There are many more complex patterns based on these some of which are described later. Table 2.1 gives the order of the key developments in Conway's Game of Life relevant to in this paper.

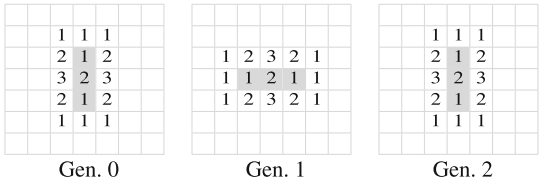

Fig. 2.1 Blinker. A period two oscillator made up of three live cells. Numbers are the count of live neighbours

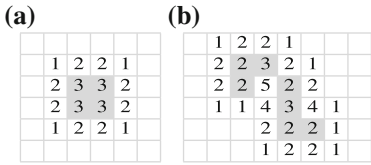

Fig. 2.2 Still life patterns do not change from one generation to another. The numbers show the count of live neighbours. **a** Block, **b** Eater

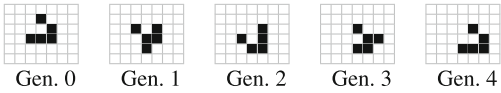

Fig. 2.3 Five generations of the glider

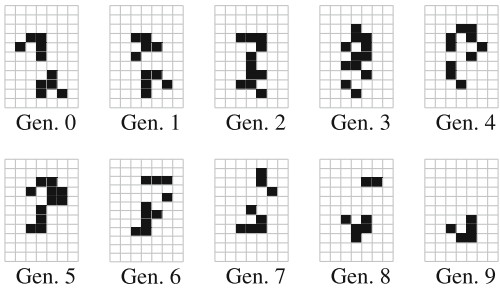

Fig. 2.4 The kickback reaction in one generation steps. A glider is reflected 180° by another glider

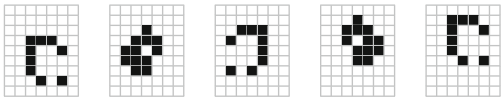

Fig. 2.5 The generations of the light weight spaceship (LWSS), a pattern which reproduces itself with an orthogonal displacement every four generations

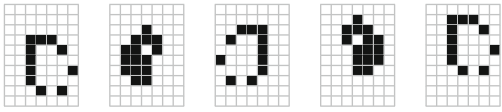

Fig. 2.6 The generations of the medium weight spaceship (MWSS)

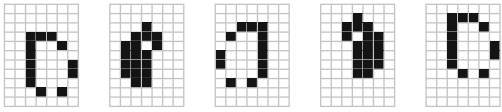

Fig. 2.7 The generations of the heavy weight spaceship (HWSS)

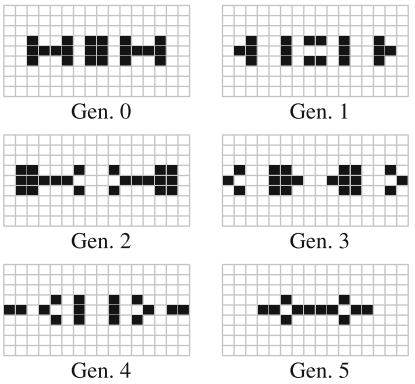

Fig. 2.8 Six generations of the period fifteen pentadecathlon oscillator showing the spark. The useful little pattern that separates from the main oscillator

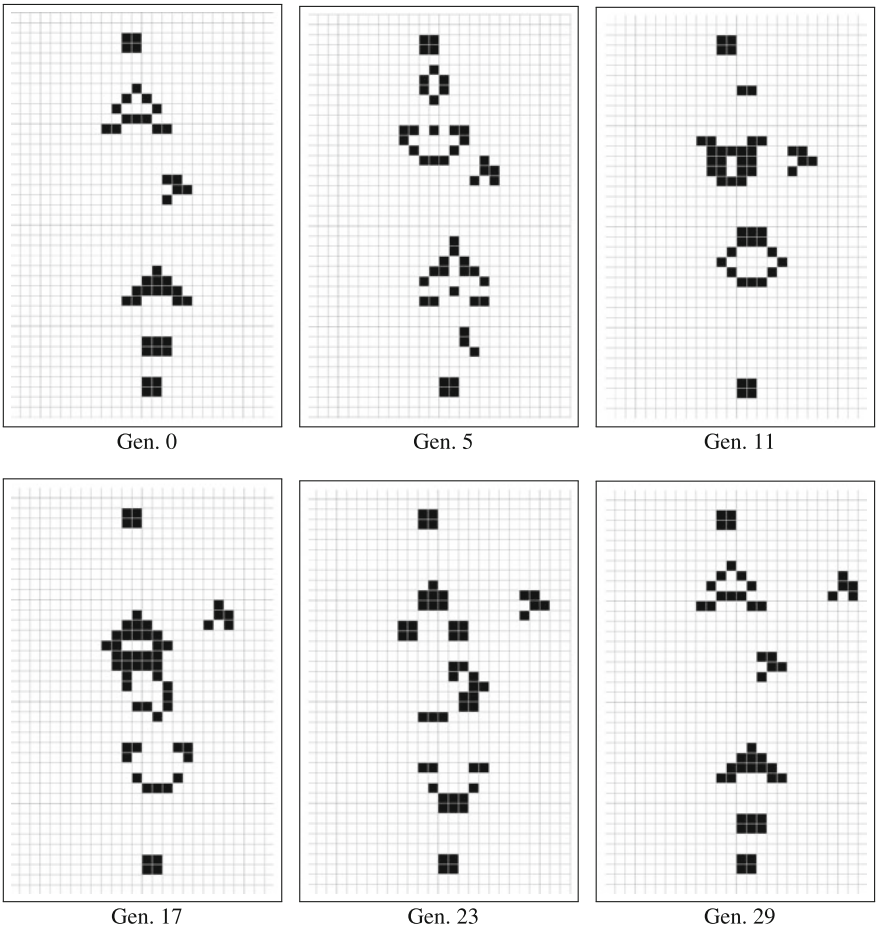

Fig. 2.9 30 generations of the Gosper gun in steps of six generations

Table 2.1 Relevant key developments in Conway's Game of Life

Date	Event
1970	Game of Life proposed by Conway in October [1]
1970	Gosper gun found in November, Fig. 2.9
1982	Winning ways proof of universality [5]
mid 80s	Buckingham and Niemiec's Adder, Sect. 3.2
1990	Dean Hickerson's Sliding Block Memory, Sect. 3.3
2000	Turing Machine Built, Chap. 4
1996	Stable Reflector Found by Paul Callahan, Sect. 3.5.2
2009	Paul Chapman's Counter Machine, Sect. 3.4
2010	Universal Turing Machine Built, Chap. 5
2010	Spartan Universal Computer-Constructor, Sect. 3.6

2.2 Turing Machines

The Turing machine is a mathematical concept invented by Alan Turing in 1936 to probe the limits of computability which culminated in the Church Turing Thesis [6]. This showed the equivalence of three different formal definitions of computability. The definition of computability using a Turing machine is now generally regarded as the clearest statement of computability, in simple terms "everything algorithmically computable is computable by a Turing machine".

The power of the Turing machine comes from its simplicity. It is designed as the simplest conceptual computing machine. It has a fixed program and an infinite storage medium in the form of a tape. It starts with its input data written on the tape and ends with its output on the tape. It has no other communication with the outside world. It is possible that the machine never stops. Despite this simplicity Turing designed a universal Turing machine, see Sect. 2.2.2.

2.2.1 Turing Machine Structure

A Turing machine consists of a finite state machine which interacts with an infinite date storage medium. The data storage medium takes the form of a unbounded tape on which symbols can be written and read back via a moving read/write head. The symbols which can appear on the tape must be members of a finite alphabet. One of these symbols is the blank symbol which initially populates all the tape except for a finite section.

The program of Turing machine is the finite state machine. This is effectively a lookup table with two indices. One index is the symbol read from the tape and the other is the machines 'state'. The state is held in the Turing machines internal memory and must be a finite value.

The Turing machine's read/write head moves along the tape in steps. At each step it reads a symbol from the tape and uses this together with its internal state to calculate a symbol to write in its place and to decide which way to move the read/write head. The cycle then repeats after updating the internal state unless the machine decides to stop.

The operation of the machine is completely determined by a table which gives for each combination of input symbol and internal state:

- The symbol to write.
- The new internal state.
- The direction to move the read/write head.
- Whether to halt or continue.

2.2.2 Universal Turing Machines

A universal Turing machine U is a Turing machine which takes as its input a description of another Turing machine T and a description of T's initial Tape. U will leave on its tape a description of the output that T would have produced. Turing first described his universal Turing machine in his 1936 paper [7].

U is said to be universal because there exists a T which performs the equivalent calculation of any Turing machine A that meets the description of a Turing machine in Sect. 2.2.1.

There are two practical issues to overcome in showing that there exists a T equivalent to any A the solution to these shown by Minsky [8] is:

- U's tape must contain a description of T. It is awkward for U to have a description of T's tape when this is infinite in both directions. Therefore T will have a tape which is finite in one direction and infinite in the other, albeit with a finite none blank pattern on it.
- A can have any finite number of symbols in its alphabet. T's alphabet must be known by U.

To cover the first point we note that for every A with a tape which is infinite in both directions there exists a T which is equivalent except that it has a tape which is only infinite in one direction. This can easily be arranged by considering T's tape as A's tape folded in half. T will simulate A's tape by grouping three symbols together.

- One to hold a symbol on A's tape going towards infinity on the left.
- One to hold a symbol on A's tape originally going towards infinity on the right, but folded over and now going towards infinity on the left.
- One for space to hold a symbol marking the middle of A's tape at the fold on T's tape.

It is then a trivial matter for T to have two sets of states, both equivalent to A, one for each half of A's tape. Both sets will have extra states for each of A's state transitions

to make the extra movements over T's tape and swap directions at the end of T's tape.

To cover the second point we note that for every A with an alphabet size n there exists a T which is equivalent except that it has an alphabet size two. This can be achieved by using several symbols on T's tape to code one symbol on A's tape. Each of A's state transitions would be replaced by a small number of transitions in T for it to recognise the symbol and write the correct symbol in its place and move the read/write head the correct amount in the correct direction.

Some very small universal Turing machines have be designed; the smallest rely on mapping the functionality of the machine T first into a tag machine as described by Minsky [8] and then mapping that onto the tape of machine U. Minsky [8] described a four symbol seven state universal Turing machine in this way. The smallest weakly universal machine on record is due to Wolfram [2]. It is said to be weakly universal as it requires an initial tape consisting of infinite repeated patterns on either side of the finite pattern representing the data. It is described in Sect. 3.8.2. The machine of Rogozhin [9] has a strong claim as the smallest strongly universal machine as it only requires an initial tape which has a finite pattern in an otherwise blank tape. See Sect. 3.7 for a description of this machine.

2.2.3 Example Turing Machine

In this section we explain how a Turing machine works using as the example the Turing machine implemented in Conway's Game of Life described in Chap. 4.

We will use as an example the Turing machine used in the Game of Life pattern [10]. This machine doubles the length of a string of a particular symbol on the tape. The tape has alphabet $A = \{'0', '1', '2'\}$ and has three states $S = \{S0, S1, S2\}$. The tape looks like Fig. 2.10a when it starts. ⇑ marks the position of the read/write head with the current state shown below. It will finish with twice as many *'1'* symbols as shown in Fig. 2.10b. Table 2.2 shows a list of these transitions.

The operation of the machine can be shown clearly by means of a state transition diagram. The diagram for this example is shown in Fig. 2.11. Each state is represented by a hexagonal box with the state name (*S0, S1 or S2* in this case) written inside the box. Arrows from one state box to another represent state transitions. The symbol at the base of the arrow represents the symbol read from the tape which triggers this

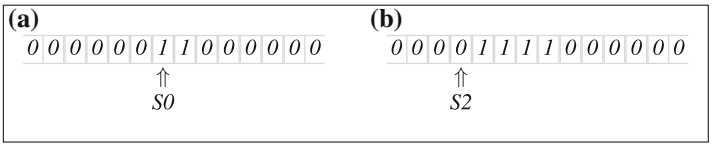

Fig. 2.10 The string doubler's TM tape. **a** Initial tape, **b** Final tape

Table 2.2 Symbol string doubler's transition list

State	Symbol	Next state	Next symbol	Direction
*S*0	*0*	*S*2	*0*	⇐
*S*0	*1*	*S*1	*2*	⇐
*S*0	*2*	*S*0	*2*	⇒
*S*1	*0*	*S*0	*2*	⇒
*S*1	*2*	*S*1	*2*	⇐
*S*2	*0*	Halt	*0*	⇐
*S*2	*2*	*S*2	*1*	⇐

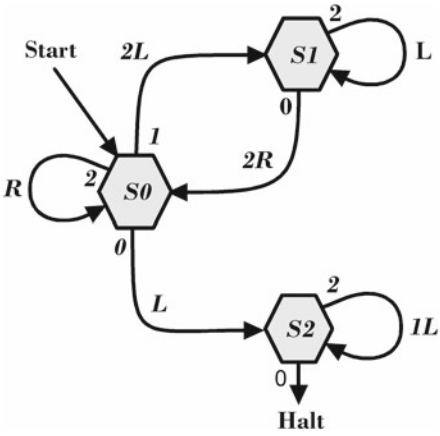

Fig. 2.11 The string doubler's TM program

transition. The symbol half way along the arrow represents the symbol written to the tape during this transition and the direction to move after writing the symbol.

An arrow may loop back to the same state indicating no change of state. If the symbol to write is the same as the symbol read, it is not shown on the arrow to reduce clutter in the diagram. No symbol at the base of an arrow indicates any other symbol.

The machine starts in state *S0* with the read/write head over the first *'1'* symbol of the string to double. If there is a *'1'* symbol on the tape it will change state to *S1* and replace the symbol *'1'* with *'2'* symbol. The symbol *'2'* is a temporary mark replacing *'1'* symbols that have been processed. The read/write head is moved left as it enters state *S1*. State *S1* performs the job of finding a blank part of tape to use as the double of the symbol found. This will be to the left of the original string of *'1'*s. When a blank part of tape is found (symbol *'0'*), the machine changes to state *S0* replacing the *'0'* with a *'2'* and moving right. State *S0* now performs the task of finding the next *'1'* to the right of the current position. It will skip over any *'2'* symbols it finds and we expect at least one of these at this stage. If a *'1'* symbol is found the machine changes to state *S1* as before. This time we expect to skip some *'2'* symbols in state *S1* to find a blank part of tape.

This sequence will continue until all the *'1'* symbols in the string have been changed to *'2'* symbols, with one blank also changing to a *'2'* symbol for each of these. The machine will be in state *S0* and read the blank symbol *'0'* to the right of the last of the original *'1'* symbols. This will trigger a change to state *S2*. This state simply moves the read/write head left and changes all the *'2'* symbols into *'1'* symbols. It will stop when it reads a *'0'* symbol. The tape will then look like Fig. 2.10b.

There are a number of Turing machine simulators available on the Internet. The author used one due to Britton [11]. This simulator requires the definition of the finite state machine to be in the form of the list of state transitions. The list of the transitions for this example is shown in Table 2.2. This simulator treats halt as a state. It therefore performs the full state transition action into this state including moving the read/write head. It also has a special symbol '_' for a blank part of tape replacing the *'0'* used in this chapter. Figure 2.12 shows a screen shot of this simulator after completing the example program.

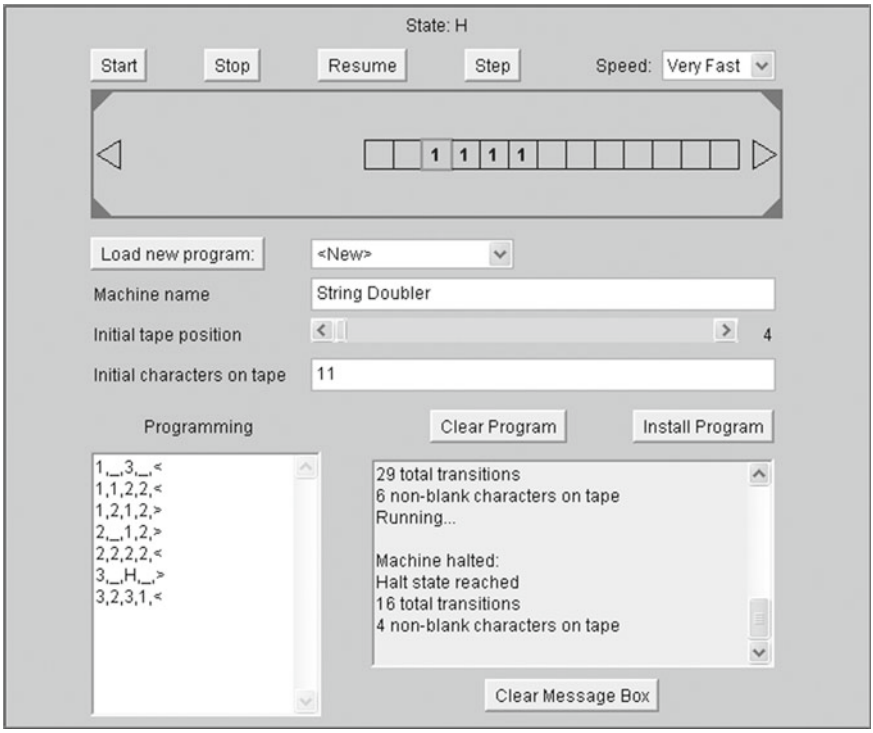

Fig. 2.12 Screenshot of the Turing machine in Fig. 2.11 being simulated by Britton [11], this simulator numbers its states from one so for this they have been renumbered 1–3

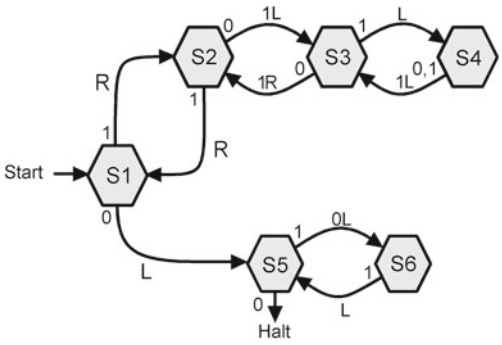

Fig. 2.13 A two state version of the Turing machine program shown in Fig. 2.11

Two symbol Turing machines are needed as the specific machine to be run by a universal machine. Figure 2.13 shows a two symbol version of the machine in Fig. 2.11 which is used for this purpose below. The two symbol version was created from the three symbol version by coding the three symbols as pairs two symbols ($0 \rightarrow 00$, $1 \rightarrow 01$, $2 \rightarrow 10$) and adding extra states to make the result equivalent. A straight forward process.

2.3 Counter Machines

A counter machine is an abstract computer like machine used as a mathematical tool to probe the limits of computability. It is equivalent to a Turing machine in its ability. A description can be found in [12]. The operation of a counter machine was described by Minsky [8] and is sometimes known as a Minsky Register Machine. It consists of a finite number of counters controlled by a simple program consisting of a list of labelled instructions. These instructions are executed one after another in the manner of a conventional computer, except that the instruction set is very small and the counters can theoretically hold any positive number however large. A typical instruction set is:

id INC *c next*	Increment counter *c* then go to instruction labelled *next* *id* is this instruction label
id DEC *c next onZero*	If counter *c* is zero, go to instruction labelled *onZero*, otherwise decrement counter *c* and go to instruction *next* *id* is this instruction label
id HLT	Halt. *id* is this instruction label

The following is a program for a counter machine to add the contents of counter *c1* to counter *c2* leaving counter *c1* at zero.

01 DEC *c1 02 03*
02 INC *c2 01*
03 HLT

Paul Chapman designed a counter machine pattern for the Game of Life which is described in Sect. 3.4.

2.4 Universality of the Game of Life

Conway provided proof of the universality of the Game of Life in [5]. He showed that the stream of gliders produced by the Gosper gun (Fig. 2.9) can be used to pass information from one place to another. He showed that simply by using collisions between glider streams it is possible to make all the necessary logic to construct a computer with finite storage.

In order to establish universality a computer with infinite storage is required. The method proposed by Conway was to use a counter machine. Counter machines are described in Sect. 2.3.

A counter machine can simulate a Turing machine by encoding a Turing machines tape onto two counters. One representing the contents of the tape on the left of the Turing machine's read/write head and one for the contents of the tape to the right of the read/write head. The symbol under the read/write head is part of the computation process.

It was shown by Minsky [8] that for any Turing machine which uses more than two symbols on its tape there is an equivalent Turing machine which has just two symbols as described in Sect. 2.2.2.

Let these two symbols be zero and one then the tape on both sides of the read/write head can be represented by binary numbers formed by these symbols with the least significant bit closest to the read/write head. The operation of moving the read/write head will require that one of these numbers to be divided by two and the other multiplied by two. The remainder of the division is the symbol under the new position of the read/write head. The symbol to write in this cycle is added to the number which is doubled. These operations can easily be performed by a small number of counters and some very repetitive, but finite programing. It will work for numbers of any size, i.e. it can simulate an infinite tape.

Conway noted that the value of a counter can be represented by distance. He showed that a block pattern Fig. 2.1 could be shifted either forward or back along a diagonal by suitable salvoes of gliders Fig. 2.3. The distance of the block from its base can be used to represent the counter value. He also showed that it was possible to detect when the block was in its base position and therefore that the counter held the value zero. This is all that is needed to construct a counter for a counter machine which can store any value because Conway's Game of Life has theoretically got infinite space to move the block into.

The primitive parts that Conway specified were:

- The basic patterns shown in Sect. 2.1 including the block, the eater and Gosper gun.
- NOT gate. Made from a 90° collision between glider streams.
- AND gate. Made from a sampling glider stream which collides with two input streams at 90°. The second input stream after the collision becomes the output.
- OR gate. Made from inverting a sampling glider stream after it has collided with two input streams at 90°.
- Stream thinner. A method of using the kickback reaction to change two period 30 glider streams into two period 60 glider streams. A period 30 glider stream as produced by the Gosper gun can not cross the path of another period 30 glider stream without collisions while a period 60 glider streams can. This provides a method of routing signals freely.
- Side tracking. This uses a pairs of kickback reactions to alter the path of a thinned glider stream diagonally by one cell position. This allows positioning of gliders close together which is required for the salvoes of gliders moving to block of the counter.
- Stream duplicator. Designed by using one in every ten gliders of the Gosper gun stream to code information and the others positions in the stream to make copies of the information gliders during the duplicating process. This also overcomes the routing problems as the full period 30 glider streams can not cross without collisions.

References

1. Gardner, M.: Mathematical games: the fantastic combinations of John Conway's new solitaire game 'life'. Sci. Am. **223**, 120–123 (1970)
2. Wolfram, S.: Universality and complexity in cellular automata. Physica **10D**, 1–35 (1984)
3. Cook, M.: Universality in elementary cellular automata. Complex Syst. **15**(1), 1–40 (2004)
4. Wolfram, S.: A New Kind of Science. Wolfram Media Inc., Champaign (2002)
5. Berlekamp, E., Conway, J., Guy, R.: What is life, chapter 25. Winning Ways for Your Mathematical Plays, vol. 2. Academic Press, London (1982)
6. Kleene, S.C.: Introduction to Metamathematics. North-Holland Publishing Company/ Wolters-Noordhoff Publishing, Amsterdam (1952)
7. Turing, A.M.: On computable numbers, with applications to the Entscheidungs Problem. Proc. Lond. Math. Soc. **2**, 230–265 (1937)
8. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)
9. Rogozhin, Y.: Small universal Turing machines. Theor. Comput. Sci. **168**(2), 215–240 (1996). ISSN 0304-3975
10. Rendell, P.: A simple universal Turing machine for the game of life Turing machine, chapter 26. In: Adamatzky, A. (ed.) Game of Life Cellular Automata, pp. 519–545. Springer, London (2010)
11. Britton, S.: Java Applet Turing Machine Simulator. http://ironphoenix.org/tril/tm/ (2009)
12. Moore, C., Mertens, S.: The Nature of Computation. Oxford University Press, Oxford (2011)

http://www.springer.com/978-3-319-19841-5

Turing Machine Universality of the Game of Life

Rendell, P.

2016, XV, 177 p. 141 illus., Hardcover
ISBN: 978-3-319-19841-5