A Kinematic Turing Machine

William M. Stevens

Department of Physics and Astronomy, Open University, Walton Hall, Milton Keynes, MK7 6AA, UK
william@stevens93.fsnet.co.uk,
WWW home page: http://www.srm.org.uk

Abstract. A Turing Machine in a three dimensional discrete space environment containing movable cubic parts is described. All of the cubic parts are identical in function. The only function that a part performs is to move a neighbouring part by one unit. Parts can be connected to neighbouring parts. When one part moves, parts that it is connected to also move. An example program for the Turing Machine is given.

1 Introduction

The earliest computers were mechanical. Babbage's analytical engine [2] consisted of sets of cogged wheels which stored decimal numbers and which could be engaged by a mechanism able to perform arithmetical operations on the numbers represented by the angular position of the wheels. Züse used interlocking sliding bars to implement boolean operations in his mechanical computers [14].

Recently, there has been a revival of interest in mechanical computing, partly out of a desire to explore the relationship between computation and physics, and partly because it may be possible to build nanoscale computing devices using mechanical logic. Fredkin and Toffoli described a model of computation based on elastic collisions between identical balls [4]. Merkle described a model based on sliding, interlocking rods [7].

This paper is motivated by a desire to understand what kinds of mechanical interactions are required in mechanical computers. An attempt is made to model the kinematic behaviour of a simple system without modelling any mechanism that gives rise to the kinematic behaviour in question. The model used can be classified as a 'kinematic automaton'. It is based in a 3D cellular space, where each cell can contain a cubic part. Collections of connected parts can move together like a rigid body. From such collections, mechanisms can be made.

A Turing Machine can be implemented in this system, and is presented in this paper. The Turing Machine presented has some similarities to that described by Laing in [6]. The most notable similarity is the way in which state transitions are implemented. In a state diagram, state transitions can be represented as arrows that lead from one state to another. In [6] and in the system described here, state transitions are embodied as paths that run from one state to another that get followed by the Turing Machine.

All mechanisms are made from identical component parts. This scheme arose during research into self-replication and automatic construction. If one is interested in simplifying the process of automatically constructing a computer from a set of component parts, then designing that computer using a single type of component part is advantageous. Making a computer that operates on the same type of component from which it is made is of interest from the perspective of self-replication because a self-replicating machine must be able to process the same kinds of parts that it is built from.

It is advised that this paper be read in conjunction with the supplementary material that is available for simulating the Turing Machine, as described in section 7.

2 A kinematic simulation environment

A three dimensional discrete space simulation environment that supports moveable cubic parts is used in this paper. The system is called CBlocks3D. A part called the 'SlideOn' part is used to implement a computing scheme. The SlideOn part is shown in figure 1.

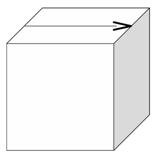

Fig. 1. The SlideOn part.

The SlideOn part will move any part positioned above it one unit in the direction that the arrow is pointing. The term 'active face' is used to refer to the face of the part that has the arrow on it.

A universe evolves through a series of states. The deterministic rule described in subsection 2.2 determines how a universe evolves from one state to another as parts act upon other parts, giving rise to movement.

2.1 Describing a state of the universe

We define six direction vectors

$$\begin{array}{rr} EAST = (1,0,0) & WEST = (-1,0,0) \\ NORTH = (0,1,0) & SOUTH = (0,-1,0) \\ FRONT = (0,0,1) & BACK = (0,0,-1) \end{array}$$

Let D denote the set of these vectors

$$D = \{EAST, WEST, NORTH, SOUTH, FRONT, BACK\}$$

The primary axis of a part is used to refer to a vector that lies on the line from the centre of the part to the centre of the active face of the part. The secondary axis of a part is the direction in which the arrow on the active face points. E.g., in figure 1 the primary axis points up the page ($NORTH$) and the secondary axis points to the right of the page ($EAST$).

We define the function

$$\text{opposite}((x, y, z)) = (-x, -y, -z)$$

The 3-tuple $((x, y, z), A, B)$ completely describes a part.

Here

$$x, y, z \in \mathbb{Z}$$
$$A \in D$$
$$B \in D \setminus \{A, \text{opposite}(A)\}$$

(x, y, z) is a position vector that specifies the location of a part.

The orientation of a part is described by specifying the direction of the main axis A and the direction of the secondary axis B . Since the main axis and the secondary axis are perpendicular, $A \cdot B = 0$.

The following defines what is meant when a direction vector V is added to a part p. If $p = ((x, y, z), A, B)$ and $V = (x', y', z') \in D$ then $p + V = ((x + x', y + y', z + z'), A, B)$.

The following notation is used to refer to elements in a 3-tuple p,

$p.position$ is the position vector (x, y, z) of p.
$p.main$ is the direction of the main axis of p.
$p.secondary$ is the direction of the secondary axis of p.

We define the neighbour relation $\|$ for parts p_1, p_2

$$p_1 \parallel p_2 \text{ if and only if } p_1.position - p_2.position \in D$$

And the joined relation $\bowtie_C$ for parts p_1, p_2 in a set of parts C

$$\begin{aligned} p_1 \bowtie_C p_2 \text{ if and only if } & p_1 \parallel p_2 \\ & \text{or there exists } p_3 \in C \text{ such that } p_1 \parallel p_3 \text{ and } p_3 \bowtie_C p_2 \end{aligned}$$

It is also useful to have a neighbour predicate which is true if p_2 is a neighbour of p_1 in direction A

$$\text{neighbour}(p_1, p_2, A) \text{ if and only if } p_2.position - p_1.position = A$$

We wish the CBlocks3D system to support collections of joined parts that move together when any one of them is pushed. Such a collection is called a ‘construct’ and a construct C must satisfy the following conditions. No two parts can have the same coordinates

$$\text{for all } p_1, p_2 \in C, \text{ if } p_1 \neq p_2 \text{ then } p_1.\mathit{position} \neq p_2.\mathit{position} \tag{1}$$

Any pair of parts in C must be joined

$$\text{for all } p_1, p_2 \in C, p_1 \bowtie_C p_2 \tag{2}$$

The active face of a part p_1 in a construct C must not lie against another part p_2 in C

$$\begin{aligned}&\text{for all } p_1 \in C \text{ there does not exist } p_2 \in C\\ &\qquad\qquad \text{such that } p_2.\mathit{position} = p_1.\mathit{position} + p_1.\mathit{main}\end{aligned} \tag{3}$$

We can define a neighbour relation and a neighbour predicate for constructs C and E based on the neighbour relation and predicate for parts.

$$C \parallel E \text{ if and only if there exists } p_1 \in C, p_2 \in E \text{ such that } p_1 \parallel p_2$$

$$\begin{aligned}&\text{neighbour}(C, E, A) \text{ if and only if there exists}\\ &\qquad\qquad p_1 \in C, p_2 \in E \text{ such that } \text{neighbour}(p_1, p_2, A)\end{aligned}$$

For constructs, it is also useful to have a joined predicate for constructs C and E in a set of constructs S

$$\begin{aligned}&\text{joined}_S(C, E, A) \text{ if and only if} \text{neighbour}(C, E, A) \text{ or there exists}\\ &\qquad\qquad F \in S \text{ such that } \text{neighbour}(C, F, A) \text{ and } \text{joined}_S(F, E, A)\end{aligned}$$

A state S is a set of constructs that satisfies the following conditions.

Constructs may not overlap.

$$\begin{aligned}&\text{for all } C, E \in S, \text{ if } C \neq E \text{ then there do not exist } p_1 \in C, p_2 \in E\\ &\qquad\qquad \text{such that } p_1.\mathit{position} = p_2.\mathit{position}\end{aligned} \tag{4}$$

If the active faces of two or more parts lie against parts in a construct C, the secondary axis of those parts must point in the same direction. This ensures that no attempt is made to slide a construct in two different directions at once.

$$\begin{aligned}&\text{for all } p_1 \in E, p_2 \in F, p_3 \in C, p_4 \in C,\\ &\qquad p_1.\mathit{main} = p_3.\mathit{position} - p_1.\mathit{position} \text{ and}\\ &\qquad p_2.\mathit{main} = p_4.\mathit{position} - p_2.\mathit{position} \text{ implies}\\ &\qquad p_1.\mathit{secondary} = p_2.\mathit{secondary}\end{aligned} \tag{5}$$

Note that in (5) it is possible that $E = F$.

2.2 Evolution of a universe

A universe evolves through an infinite sequence of states $S_0, S_1, S_2, \ldots$

S_0 completely determines subsequent states. For $n \geq 0$, S_{n+1} can be determined from S_n using the the algorithm given below. The algorithm can be subdivided into three steps:

Step 1 Work out whether the active face of a part in one construct C comes against a part in another construct E. If so, E is to be moved.

Step 2 Work out whether any constructs will be pushed by E when E moves.

Step 3 Move every construct that is to be moved.

A formal description of this algorithm follows.

Associated with every construct C in S_n is a movement set $M_{C,n} \subseteq D$

```
Step 1
For each construct $E$ in $S_n$
    For each part $p_1$ in $E$
        If there exists $p_2$ in a construct $C \in S_n$ such that
        $p_2.position - p_1.position = p_1.main$ then $p_1.secondary \in M_{C,n}$
```

```
Step 2
For each construct $C$ in $S_n$
    If $M_{C,n}$ contains an member $A$ then
        for all constructs $D$ satisfying joined$(C, D, A)$, $A \in M_{D,n}$
```

It is illegal for any $M_{C,n}$ to have more than one member. This situation would arise if (5) were violated, and also if an attempt were made to displace a construct in two different directions.

```
Step 3
For each construct $C$ in $S_n$
    If $M_{C,n}$ is empty, then $C \in S_{n+1}$
        otherwise, if $M_{C,n}$ has a member $A$ then $C + A \in S_{n+1}$
```

The result of the operation of adding the vector $A \in D$ to the construct C in step 3 is a construct defined by:

$$p \in C \text{ if and only if } p + A \in C + A \tag{6}$$

It is illegal for (4) to be violated at this point in the algorithm. It is also illegal to have $|S_{n+1}| < |S_n|$. (Without the latter constraint, two identically shaped constructs could end up occupying the same coordinates).

The ‘illegal’ conditions described above exist to prevent having to deal with conflict situations that would arise for example, if an attempt were made to slide a construct in two different directions at the same time or if an attempt were made to move two parts into the same location. Of course, rules could be written to resolve such conflict situations (as Arbib does in [1]), but since such situations do not arise in any of the mechanisms described in this paper, and since a description of these conflict resolution rules would be lengthy, we simply define these situations as illegal and rule that no mechanism should depend upon illegal states.

Note that the Turing Machine described later in this paper does not make use of the ability of constructs to shunt neighbouring constructs along (i.e. Step 2 above).

2.3 Relationship with Cellular Automata environments

Cellular automata have been widely studied and several results regarding the computational capabilities of simple CA are known. Rendell showed that Conway’s Game of Life is computation-universal [9]. Cook showed that 1D cellular automaton rule 110 can be used to implement a cyclic tag system [3].

Some kinematic automata, such as that described in [10], have a direct cellular automata implementation. CBlocks3D does not because of the possibility of connecting parts together so that they move together when pushed.

In common with cellular automata environments, CBlocks3D is based in a discrete space, divided up into identical cells.

The differences between CBlocks3D and cellular automata are listed below.

– In CBlocks3D all parts are identical and do not change with time. In CA cells can be in one of a number of states. The state of a cell can change with time.
– In CBlocks3D parts can move to neighbouring locations. In CA cells are fixed in position. Motion can be simulated by shifting a state from one cell to a neighbour.
– In CBlocks3D a part is directly affected by neighbouring parts, and can also be affected by parts that it is joined to, no matter how far away. In CA, a cell is only directly affected by the states of a finite and predetermined set of its neighbours.
– In CBlocks3D the number of parts is conserved. In CA the number of cells in a particular state may vary.

Cellular automata are excellent tools for studying the logical behaviour of a system and for exploring geometric constraints imposed by particular spaces, but are not the right tool for studying the interactions of rigid structures. The environment described here provides an extension to the cellular automata model that allows the computational properties of interacting rigid structures to be studied. A similar environment was used by Goel and Thompson in [5] to study biological self-assembly.

It is possible to imagine environments that combine the properties of kinematic automata with those of cellular automata. For example, parts in a kinematic automata could be in any one of a number of states and the state of a part could be affected by the states of neighbouring parts. A hybrid environment of this kind is used in [11], [1]. Such systems can be usefully classified as 'kinematic cellular automata'. (A term first used by Toth-Fejel in [12]).

3 Notation and Petri Net diagrams

Using the definitions given in the previous section it would be possible to use set notation to completely describe the states that a system goes through as it evolves. Such a description would give little insight into the behaviour of a system and would be a cumbersome tool to use as a means of proving a system's capabilities. In this section, a notation is introduced that can be used to describe the behaviour of a system at a higher level of abstraction than that used in the previous section.

Figures 2 and 3 show successive states S_n and S_{n+1} of a universe.

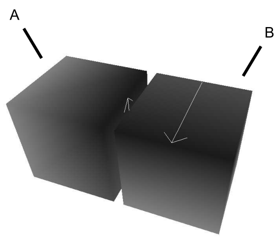

Fig. 2. A simple mechanism in state S_n. Constructs A and B are labelled.

Figure 4 is a state diagram describing the system. S_n and S_{n+1} are respresented by circles. The transition between S_n and S_{n+1} is represented by an arrow, labelled with the interaction between constructs that causes the state to change. In this case, the interaction is $A, B \to B[n]$. Meaning 'An interaction between constructs A and B causes B to move North'. The whole diagram means 'When the state of the system is S_n, an interaction between A and B causes B to move North, and the system ends up in state S_{n+1}.

For systems containing several interacting mechanisms, a method for describing interacting state diagrams is required. Petri Nets [8] are suitable for this purpose.

Consider the interacting mechanisms shown in figure 5. The behaviour of the mechanism labelled CYC can be described by the state diagram shown in figure 6. This mechanism is not influenced in any way by its neighbouring mechanism, FLIP. On the other hand, construct D in mechanism FLIP is influenced by

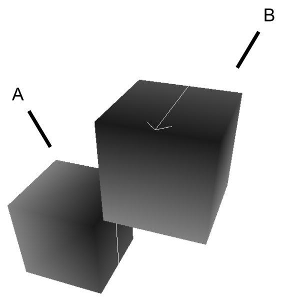

Fig. 3. A simple mechanism one time unit later in state S_{n+1}.

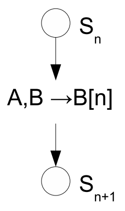

Fig. 4. State diagram for a simple mechanism.

construct B in mechanism CYC. A Petri Net for the complete system is shown in figure 7. The Petri Net can be thought of as being two state diagrams joined together, with dots to indicate the current state in each diagram. Dots move from one state to another via a transition whenever all arrows leading to a transition start from states with dots in. For more about Petri Nets see [13].

4 Turing Machines - definitions

The following formal description of a Turing Machine is used.

Set of symbols $\Sigma = \{0, 1\}$
Set of states Q
Transition function $\delta : Q \times \Sigma \to Q \times \Sigma \times \{-1, 1\}$
Starting state q_0
Halting state q_h

The state of a Turing Machine at a particular step n in its evolution is called a configuration C_n, defined as follows:

$C_n.q \in Q$ is the state of the machine at step n.
$C_n.a_1...a_t \in \Sigma^t$ are the contents of the tape at step n.

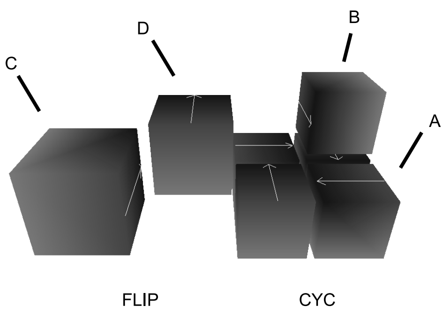

Fig. 5. Two interacting mechanisms.

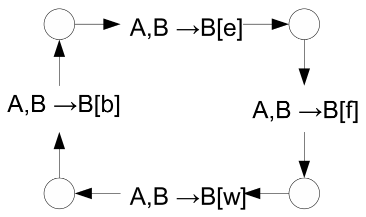

Fig. 6. State diagram for CYC mechanism.

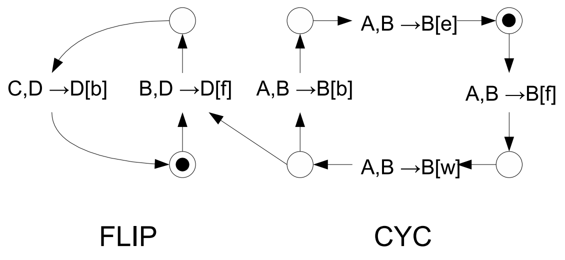

Fig. 7. Petri Net diagram for CYC and FLIP mechanisms.

$C_n.i$ is the position of the head on the tape at step n.
$C_n.a_i$ is used as an abbreviation for $C_n.a_{C_n.i}$.

The initial configuration of the machine is:

$C_0.q = q_0$
$C_0.i = 1$
$C_0.a_1...a_t$ are the initial contents of the tape.

The evolution of a Turing Machine over time is given by equation 7. Any contents of the data tape not changed by equation 7 remain unchanged from step n to step $n + 1$.

$$(C_{n+1}.q, C_{n+1}.a_{c_n.i}, C_{n+1}.i - C_n.i) = \delta(C_n.q, C_n.a_{c_n.i}) \tag{7}$$

$\delta(q, a).state$, $\delta(q, a).symbol$ and $\delta(q, a).direction$ are used when it is necessary to refer to individual elements from the tuple $\delta(q, a)$.

The Turing Machine halts when $C_n.q = q_h$, and C_n is the final configuration of the machine.

5 A Turing Machine in the CBlocks3D environment made from SlideOn parts

5.1 Overview

Figure 8 gives an overview of the Turing Machine presented in this paper.

The Turing Machine contains the following mechanisms, each of which is described in detail later on:

- The data tape mechanism DT: This implements a sequence of bits corresponding to the Turing Machine's tape.
- The program plane construct PP: This implements the transition function δ. For every $q \in Q$ there is a corresponding position of the program plane PP. Let $F : Q \to \mathbb{Z} \times \mathbb{Z}$ be the function that maps Turing Machine states to program plane positions. A state transition is carried out by moving PP from one position to another. Information encoded on PP tells the machine how to respond when a 0 bit or a 1 bit is encountered on the data tape DT. PP also contains parts that trigger the sequencer SQ via the trigger mechanism TR when a state transition has finished.
- The bit-reading mechanism BR: Interrogates the current position on the data tape DT and activates the selection mechanism SL if a 1 bit is encountered.
- The selection mechanism SL: This moves the program plane PP so as to select the second of two sets of information encoded for a particular state on PP.
- The condition-action mechanism CA: Interrogates the program plane PP and performs bit-writing and/or tape-moving actions according to information associated with the current state on PP.

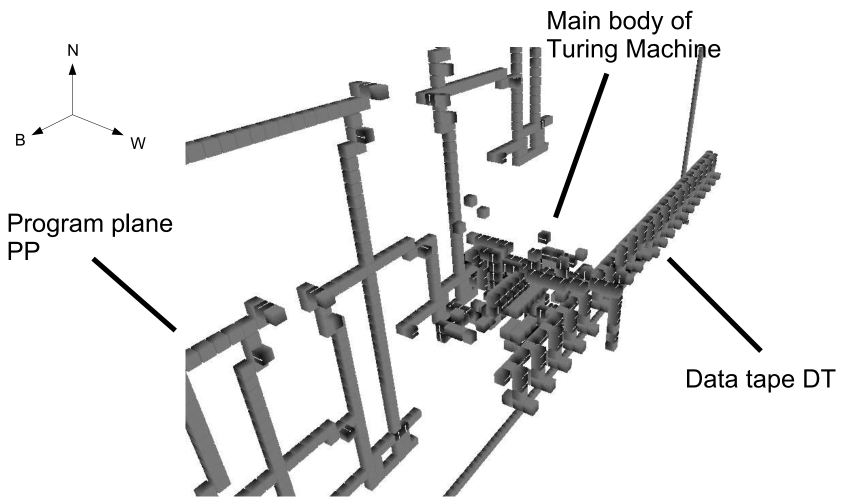

Fig. 8. Turing Machine - Overview

- The conditional-tape-moving mechanism CTM: Moves the data tape DT six units backward if activated by the condition-action mechanism CA.
- The unconditional-tape-moving mechanism UTM: Moves the data tape DT three units forward when activated by the sequencer SQ.
- The sequencer mechanism SQ: Triggers various parts of the machine in turn in order to perform a cycle of operation. Triggers UTM, then BR, then CA, then PM, then waits to be reset by the program plane PP after a state transition has occurred.
- The plane-moving mechanism PM: Follows tracks on the program plane PP to effect a transition from one state to another by moving PP.
- The trigger mechanism TR: Activated by the program plane PP when a state transition has finished in order to reset the sequencer SQ.

Section 6 contains illustrations for all of these mechanisms, along with Petri Net diagrams which concisely describe the operation of each mechanism.

5.2 Operation

Suppose that the Turing Machine is in a state corresponding to the configuration C_n. Then the z coordinate of the data tape DT is related to $C_n.i$ by $DT.z = 3 \times C_n.i + A$ where A is a constant that depends on the absolute location of the Turing Machine in space, and bit $C_n.a_p$ is represented by pin $DT.B_p$.

The main body of the Turing Machine is made from mechanisms BR, SL, BW, CTM, PM, UTM, SQ and TR and always remains stationary (apart from movements of internal mechanisms). Only DT and PP move around.

The operation of the machine is as follows:

1. The machine is in a state corresponding to a configuration C_n.
2. SQ triggers BR, CA, PM and UTM in turn.
3. BR reads the current symbol from DT. If the symbol is 0, SL does not move PP, otherwise SL moves PP four units backward.
4. CA first examines PP to determine $\delta(C_n.q, C_n.a_i).symbol$, and then sets the current symbol on DT to this.
5. CA then examines PP to determine $\delta(C_n.q, C_n.a_i).direction$ and hence which direction to move DT in. It will either leave DT as it is, or move DT six units backward.
6. PM causes PP to move along a path from $F(C_n.q)$ to $F(\delta(C_n.q, C_n.a_i).state)$
7. When PP has moved to this new position, PP resets SQ.
8. SQ triggers UTM and UTM moves DT three units forward.
9. The machine is now in a state corresponding to configuration C_{n+1}. Operation continues at step 1.

So, by the end of a cycle of operation, the following has happened:

Depending on the current location of PP (corresponding to $C_n.q$) and the position of the pin $DT.B_{C_n.i}$ at the current location on DT (corresponding to $C_n.a_i$), a symbol corresponding to $\delta(C_n.q, C_n.a_i).symbol$ has been written to DT, DT has been moved 3 places forwards or backwards, corresdponding to $\delta(C_n.q, C_n.a_i).direction$, and PP has been moved to a new position corresponding to $\delta(C_n.q, C_n.a_i).state$. This action corresponds to the abstract operation described by equation 7.

The machine can be made to halt by having a path on PP that leads nowhere. When the machine tries to move PP to a new state by following this path, the operation of the machine will cease.

6 Detailed description of mechanisms

6.1 Data Tape DT

Figure 9 shows part of the mechanism DT for representing a sequence of binary digits. Figure 10 shows the same mechanism viewed from the opposite direction. The mechanism consists of a long rod A situated against a series of pins B_x that can be individually raised or lowered with respect to the rod to represent binary 1 and 0 digits respectively. The mechanism is designed so that when the rod A is moved east or west, the pins get moved along with it so that the representation is not disturbed.

Figures 11 and 12 shows Petri Nets for the data tape DT. Note that DT has a very large number of possible states, equal to the number of possible z-coordinates of DT multiplied by the number of integers that all of the bits on DT can represent. Because of this, figures 11 and 12 use a pair of integers i, j to represent any one of a large set of possible states, where i corresponds to the number represented by all of the bits on DT (and so setting or resetting a bit corresponds to setting $i := i + 2^n$ or $i := i - 2^n$), and where j corresponds to the z-coordinate of DT.

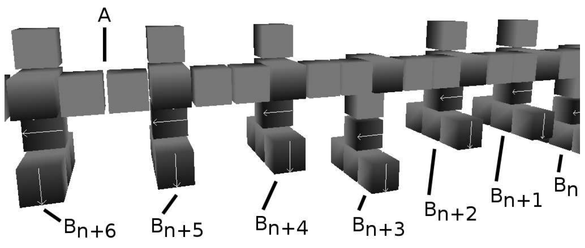

Fig. 9. The DT mechanism for representing a binary string. (View 1)

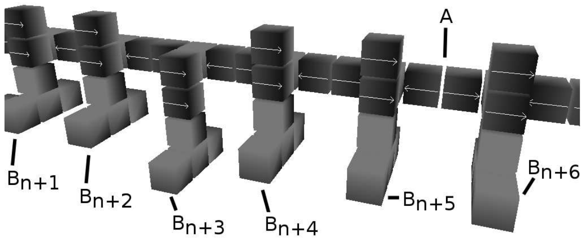

Fig. 10. The DT mechanism for representing a binary string. (View 2)

The large, dashed-outline circles in figures 11 and 12 represent states in other Petri Nets. For example, in figure 11, the dashed circle labelled with UTM_1 represents the state labelled 1 in the Petri Net for UTM (figure 24).

6.2 Bit Reading Mechanism BR

Figure 13 shows a mechanism for reading a bit on the data tape. Figure 14 shows an exploded view of the same mechanism in which the different constructs that make up the mechanism can be distinguished.

Figure 15 shows the Petri Net for the Bit Reading mechanism BR, which explains its operation. Note that this Petri Net uses an 'inhibit' arc (denoted by a line terminated with a circle). A transition that has an inhibit arc as an input cannot take place if the state from which the inhibit arc emanates is marked.

In the Petri Net for BR, the inhibit arc is used so that the network 'chooses' between the path which corresponds to a 1 bit being read from DT, and the path which corresponds to a 0 bit being read from DT.

6.3 Selection Mechanism SL

Figure 16 shows the mechanism that selects the second of two positions for a given state on the program plane, if it is activated by BR. Consider the sentence:

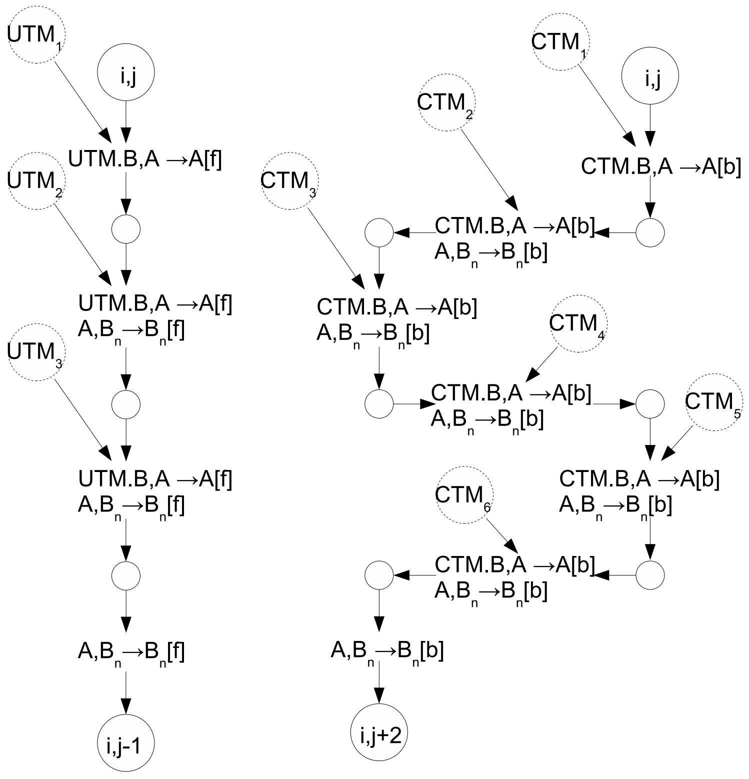

Fig. 11. Petri Net for DT. Moving back and forth.

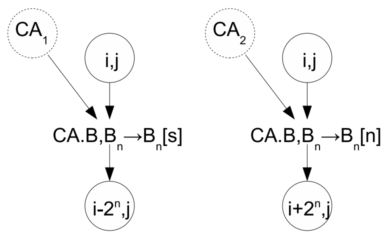

Fig. 12. Petri Net for DT. Resetting and setting a bit.

'If the Turing Machine is in state q and encounters a 0, then do ..., otherwise if it encounters a 1 then do ...'. The SL mechanism is the implementation of the word 'otherwise' in this sentence.

Figure 17 shows the Petri Net for the SL mechanism.

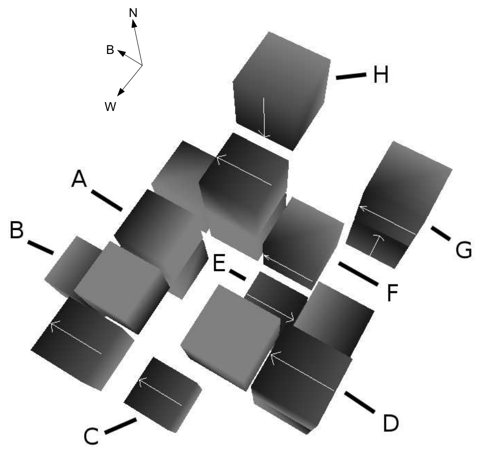

Fig. 13. The BR mechanism for detecting the state of a single bit on a data tape.

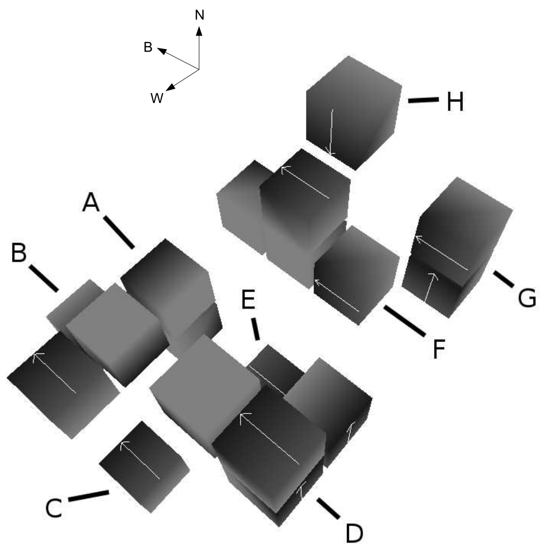

Fig. 14. Exploded view of figure 13.

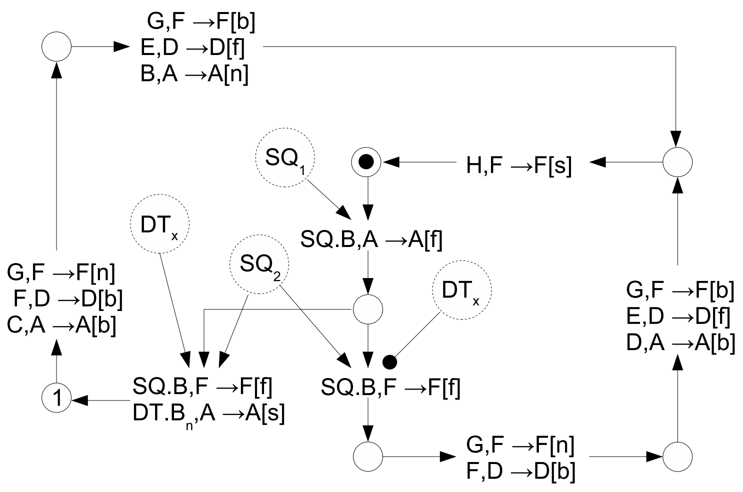

Fig. 15. Petri Net for BR.

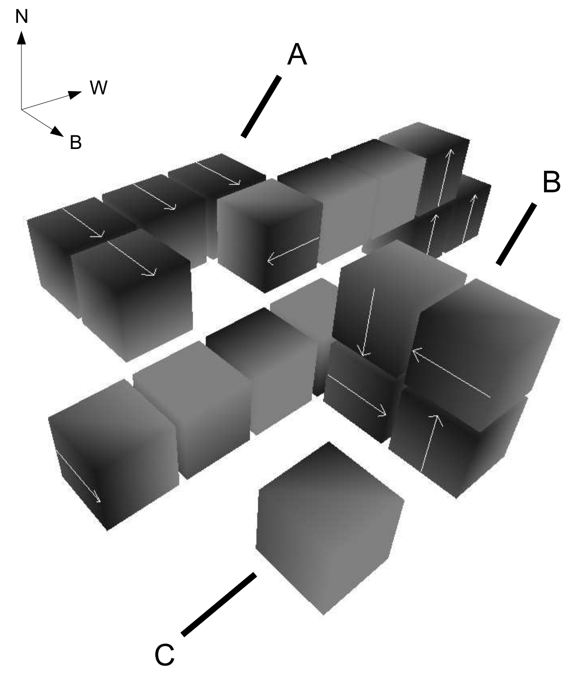

Fig. 16. The SL mechanism for conditionally moving the program plane.

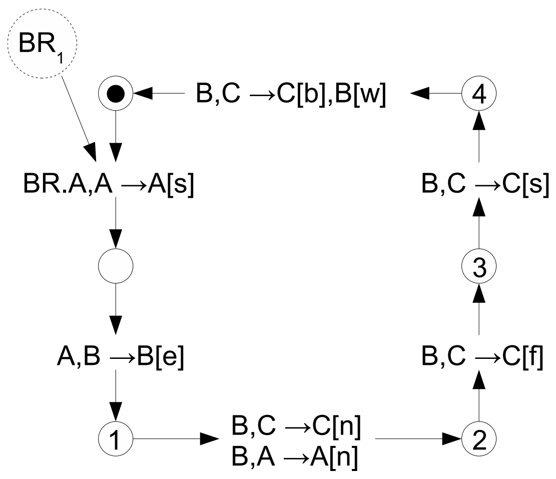

Fig. 17. Petri Net for SL.

6.4 Condition Action Mechanism CA

After the Selection Mechanism SL has ensured that the correct part of the current state on the program plane PP is in the right place, the CA mechanism interrogates that place on PP to work out whether to write a 1 or a 0 onto the data tape DT, and whether to move DT backwards or forwards.

Figures 18 and 19 show the CA mechanism. Basically it consists of two rods, one rod (labelled B) for determining the state of the 'Write a 1' bit on the program plane. The other rod (labelled A) for determining the state of the 'Advance the tape' bit on the program plane. Each rod is pressed against the program plane, and the corresponding action is triggered.

Figure 20 shows the Petri Net for the CA. The Petri Net is complex because the mechanism consists of three main constructs - the two rods A and B, and a construct G that is responsible for moving rod B back to its initial position once PP has been interrogated. Some arcs on the Petri Net that would show dependencies between the three loops in figure 20 have been omitted to avoid clutter, since they do not affect the behaviour of the net.

6.5 Conditional Tape Moving Mechanism CTM

Figure 21 shows the CTM. This mechanism is triggered by the CA when the program plane PP indicates that the data tape DT is to be moved. The CTM moves DT 6 units backwards, so that between them CTM and UTM (see the next subsection) move the DT either 3 units backwards or 3 units forwards.

Figure 22 shows the Petri Net for the CTM.

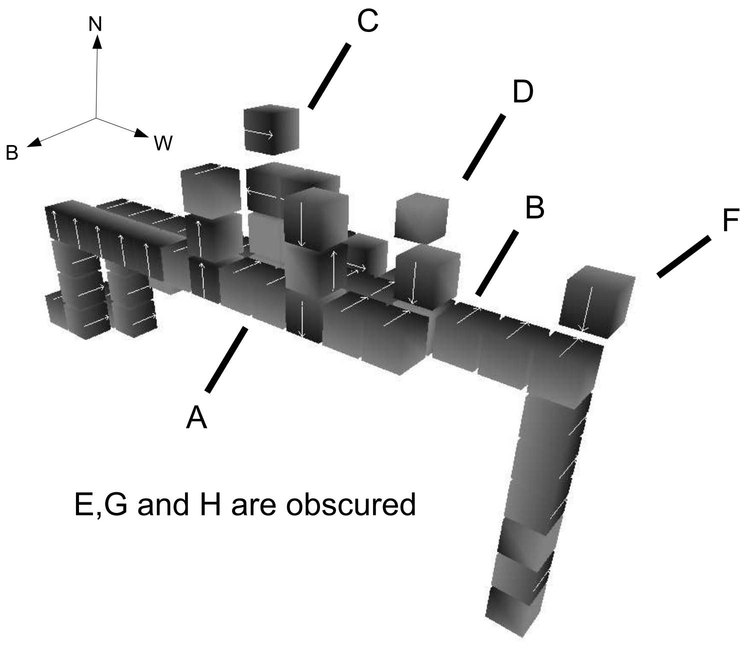

Fig. 18. The CA mechanism that performs actions depending on information encoded on the program plane.

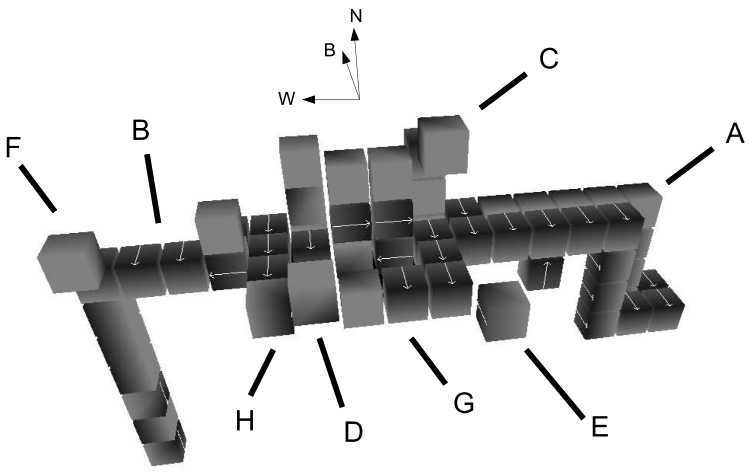

Fig. 19. A different view of the CA mechanism

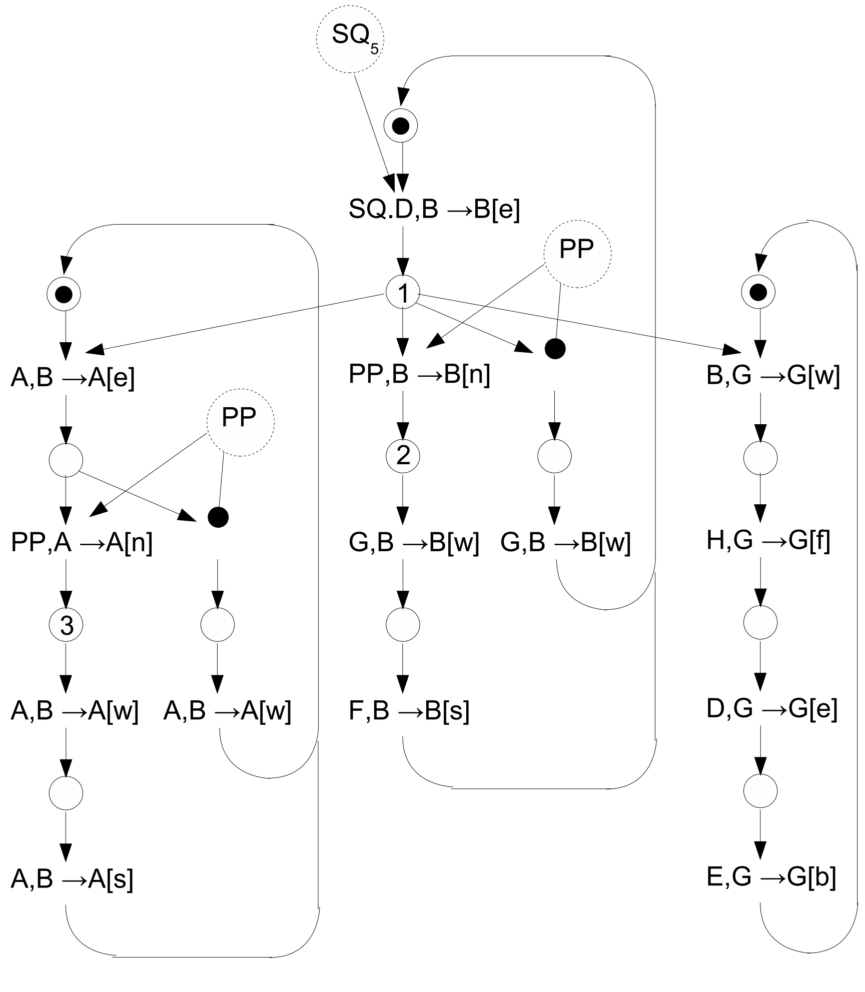

Fig. 20. Petri Net for CA.

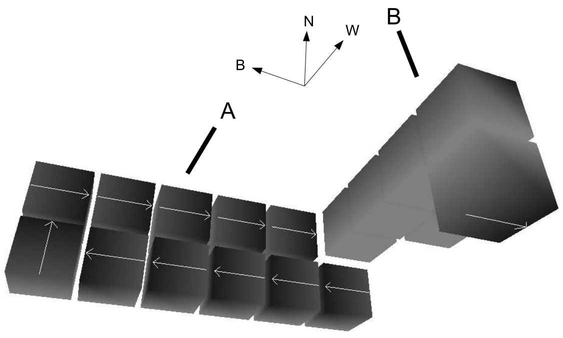

Fig. 21. The CTM mechanism

6.6 Unconditional Tape Moving Mechanism UTM

Figure 23 shows the UTM. This mechanism is triggered by the sequencer to move the DT 3 units forwards.

Figure 24 shows the Petri Net for the UTM.

6.7 Sequencer SQ

Figures 25 and 26 show the Sequencer SQ which is responsible for activating several other mechanisms: BR, CA, PM and UTM.

It can be seen that SQ is essentially a track A along which a construct B moves. In several places B encounters other constructs (not shown in figures 25 and 26) and activates the mechanisms to which they belong. Figure 27 shows the Petri Net for SQ.

6.8 Plane Moving Mechanism PM

Figures 28 and 29 show the plane moving mechanism PM. This mechanism gets activated by SQ after all actions related to reading and acting on information encoded at the current position of the program plane PP have finished. The mechanism tracks paths on the program plane PP that lead from the portion of PP that has just been examined to another state position on PP.

Figure 30 shows the Petri Net for PM. To avoid clutter in this diagram, not all PP states that have arcs leading to transitions are shown.

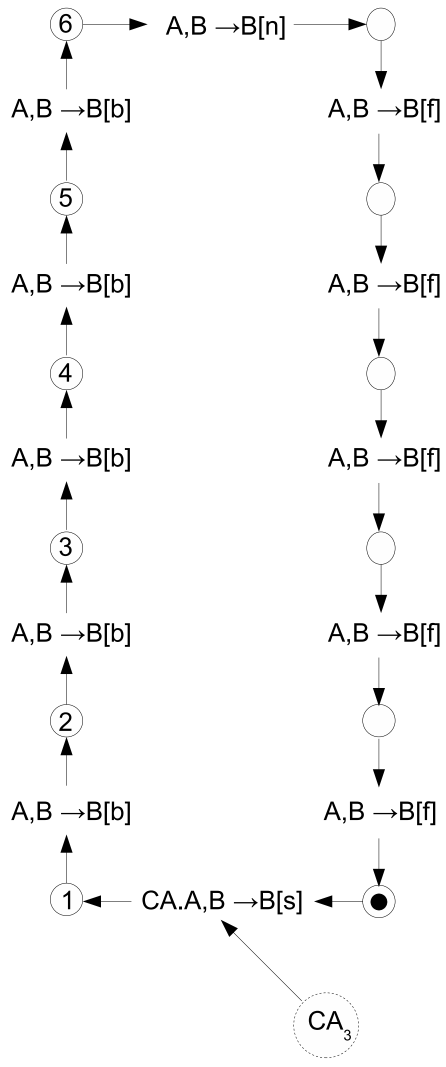

Fig. 22. Petri Net for CTM.

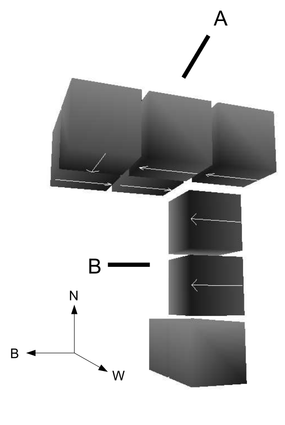

Fig. 23. The UTM mechanism.

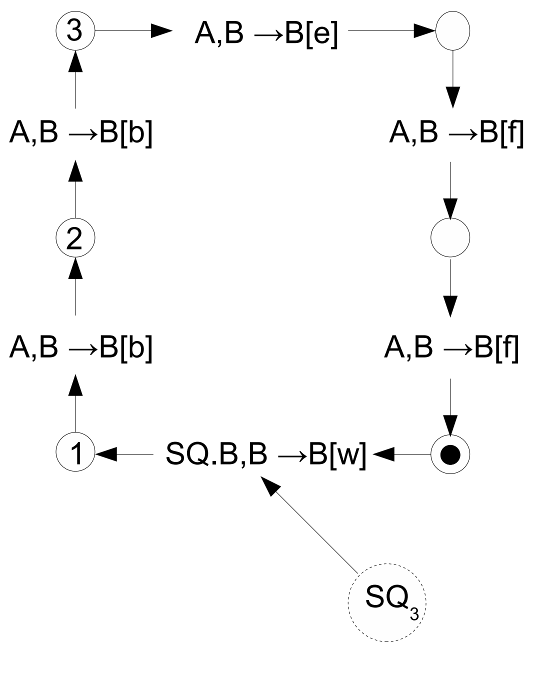

Fig. 24. Petri Net for UTM.

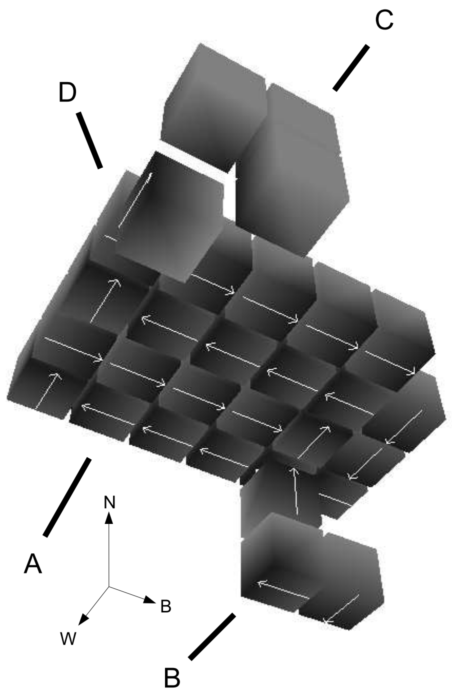

Fig. 25. The SQ mechanism.

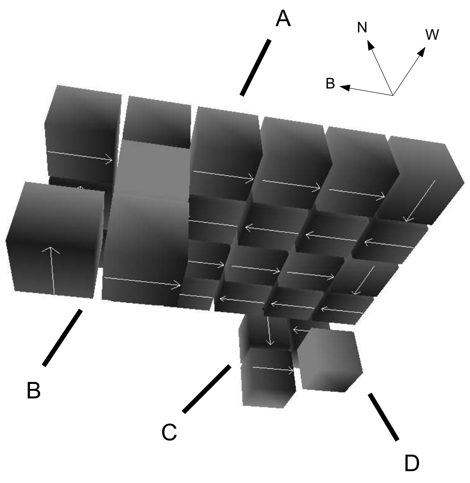

Fig. 26. A different view of the SQ mechanism.

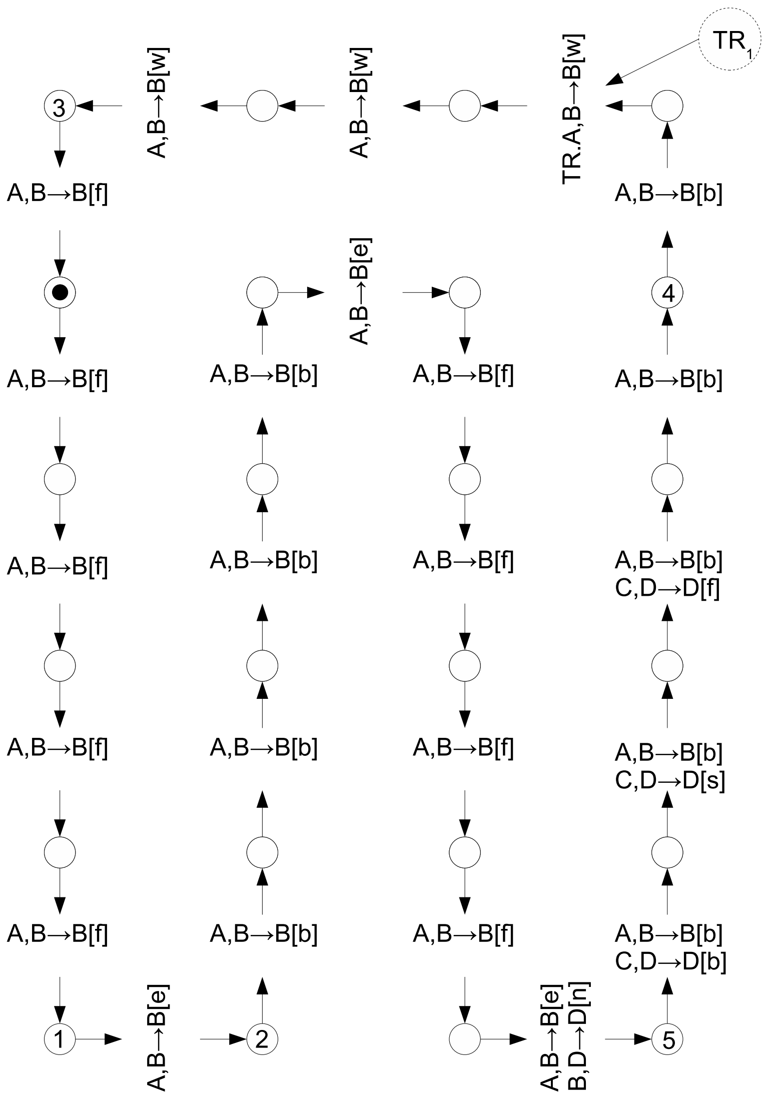

Fig. 27. Petri Net for SQ.

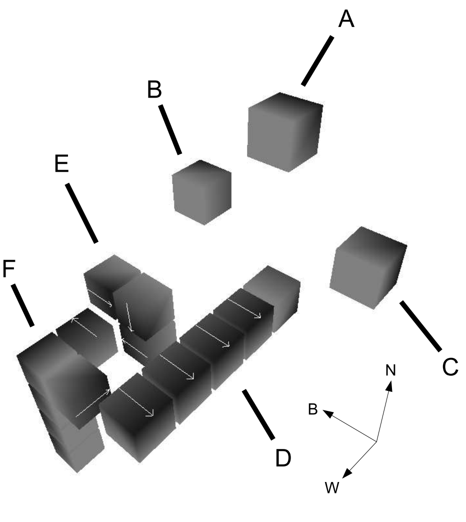

Fig. 28. The PM mechanism.

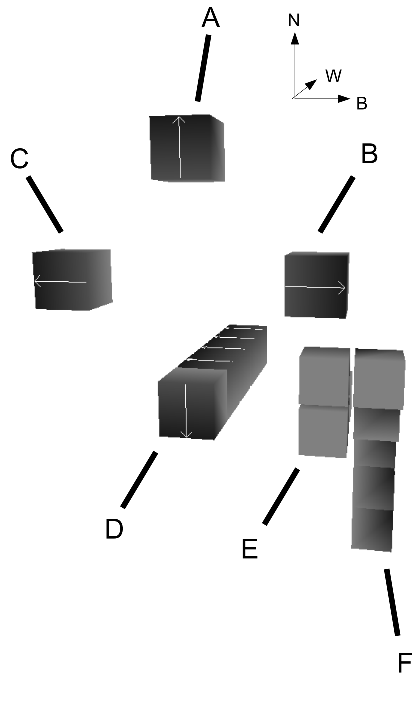

Fig. 29. A different view of the PM mechanism.

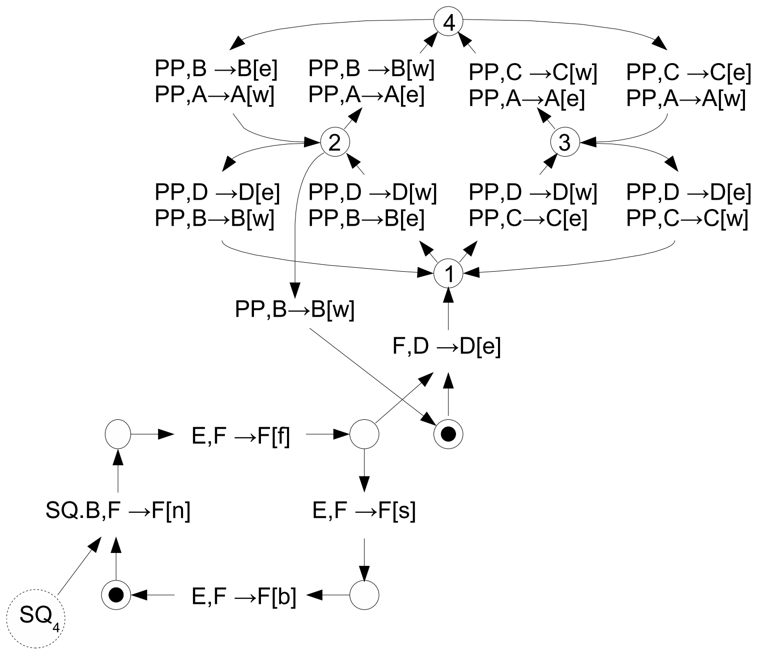

Fig. 30. Petri Net for PM.

6.9 Trigger Mechanism TR

Figure 31 shows the trigger mechanism TR that starts the sequencer when the program plane PP reaches a state position.

Figure 32 shows the Petri Net for TR.

6.10 Program Plane PP

Figure 33 shows a portion of the program plane. This corresponds to a single state in the abstract Turing Machine.

The part labelled A is the part that activates the trigger mechanism TR which in turn activates the sequencer SQ when a state position is reached. The pair of parts labelled B correspond to the two bits of information needed to tell the Turing Machine what to do when it encounters a 0 on the data tape DT. Similarly, the pair of parts labelled C tell the Turing Machine what to do when it encounters a 1 on the data tape. For B and C, a part *p* with *p.main* pointing West towards the body of the Turing Machine and *p.secondary* pointing North represents 1, and a part in any other legal orientation represents 0. In both B and C, the first (i.e. back-most) of the two bits of information tells the machine

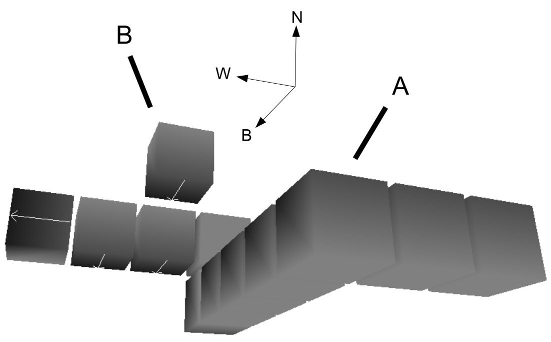

Fig. 31. The TR mechanism.

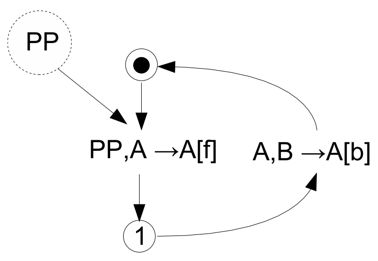

Fig. 32. Petri Net for TR.

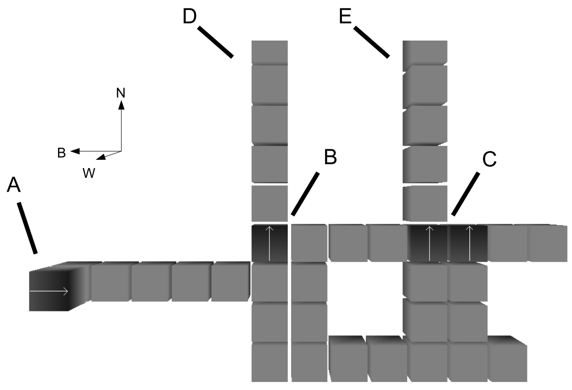

Fig. 33. The PP mechanism.

whether to move DT (1) or not (0). The second of the two bits tells the machine whether to write a 1 or a 0 onto DT. The paths labelled D and E are the paths that the plane moving mechanism PM tracks when leaving this state for another state. Which path PM follows depends on whether a 0 or a 1 was encountered on DT.

Figure 34 shows the corner of a track in PP. This figure should be examined in conjunction with the Petri Net for PM in figure 30 to see how PM and PP negotiate a corner.

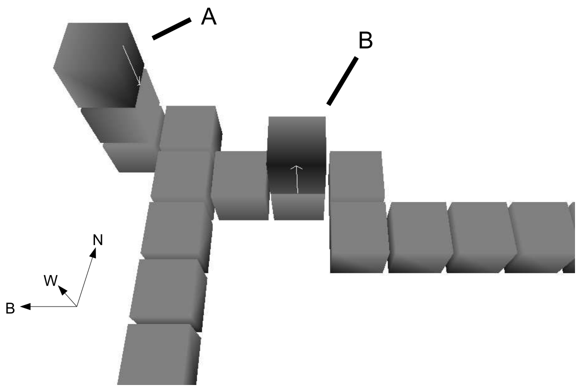

Fig. 34. The PP mechanism.

Figures 35 and 36 show partial Petri Nets for the program plane PP. (The complete Petri Net would be large and not instructive, so is omitted).

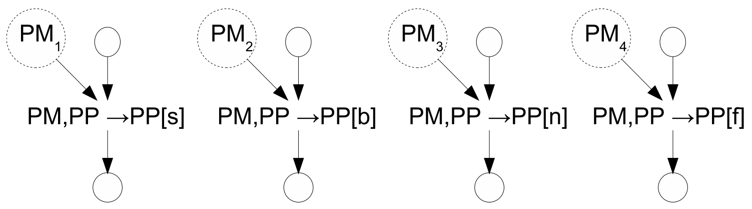

Fig. 35. Petri Net for PP. PP being moved by PM.

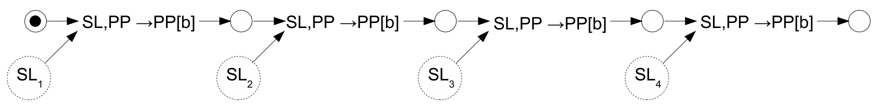

Fig. 36. Petri Net for PP showing the action of SL on PP.

7 An example

Supplementary material for this paper can be found at:

http://www.srm.org.uk/downloads/kinturing.zip

This file contains software for simulating the CBlocks3D environment and a file containing an example Turing Machine that contains a program plane for multiplying two unary numbers together, and a data tape containing the numbers 2 and 3. After several minutes of simulation time, the machine will halt with a number 6 on the data tape.

8 Discussion

The most notable aspect of the Turing Machine presented here is that it is made from a single type of part and depends upon only one type of sliding kinematic interaction.

The computational properties of a similar two dimensional kinematic system were investigated in [10]. From the perspective of physical implementation, the CBlocks3D system is much less attractive than the system in [10]. This is partly because CBlocks3D is three dimensional, and it is difficult to engineer 3D substrates with properties different from the ones that nature provides us with. (Unlike in two dimensions, where a surface can be modified two produce a range of properties). Also, in [10] there is an action and reaction between parts, but in CBlocks3D parts are able to slide neighbours without moving themselves.

The CBlocks3D universe as described here does not have construction-closure. i.e. mechanisms can exist in the CBlocks3D universe which cannot be constructed by any mechanism in the universe. The main reason for this is that there is no 'connecting' operation by means of which two constructs may become one.

It may be possible to define 'connecting' and 'disconnecting' operations that occur when two constructs are pushed together or pulled apart by SlideOn parts with opposing secondary axes. Such a system might have construction closure, and it might be possible to devise a programmable-constructor based self-replicating machine in this system.

References

1. Arbib, M.A.: Theories of Abstract Automata. Prentice-Hall, Englewood Cliffs, New Jersey (1969) 355–361
2. Babbage, C.: Passages from the Life of a Philosopher. Chapter 8: Of The Analytical Engine Longman, Green, Longman, Roberts and Green (1864)
   http://www.fourmilab.ch/babbage/lpae.html
   Cited on 13 May 2007
3. Cook, M.: Universality in Elementary Cellular Automata. Complex Systems **15** (2004) 1–40
4. Fredkin, E., Toffoli, T.: Conservative Logic. J. Theo. Phys. **21(3,4)** (1982) 219–253
5. Thompson, R.L., Goel, N.S.: Movable Finite Automata (MFA) models for biological systems. I: Bacteriophage assembly and operation. J. Theor. Biol. **131(3)** (1988) 351–385
6. Laing, R.A.: Some alternative reproductive strategies in artificial molecular machines. J. Theor. Biol. **54** (1975) 63–84.
7. Merkle, R.C.: Two Types of Mechanical Reversible Logic. Nanotechnology **4** (1994) 114–131
   http://www.zyvex.com/nanotech/mechano.html
   Cited on 13 May 2007
8. Petri, C.A.: Kommunikation mit Automaten. PhD Thesis, University of Bonn. (1962)
9. Rendell, P.: Turing Universality of the Game of Life. Collision-Based Computing (2002) 513–539
10. Stevens, W.M.: Logic circuits in a system of repelling particles. From Utopian to Genuine Unconventional Computers, Luniver Press, Frome, UK (2006) 157–182
11. Stevens, W.M.: Simulating Self Replicating Machines. J. Intelligent and Robotic Sys. **49(2)** (2007) 135–150
12. Toth-Fejel, T.: LEGO(TM) to the Stars: Active Mesostructures, Kinematic Cellular Automata, and Parallel Nanomachines for Space Applications. The Assembler **4(3)** (1996)
    http://www.islandone.org/MMSG/9609lego.htm
    Cited on 13 May 2007
13. Wikipedia article on Petri Nets.
    http://en.wikipedia.org/wiki/Petri_net
    Cited on 13 May 2007
14. Rojas, R.: Konrad Züse's Legacy: The Architecture of the Z1 and Z3. IEEE Annals of the History of Computing. **19(2)** (1997) See also: http://www.epemag.com/zuse/
    Cited on 13 May 2007