Simulating Self-Replicating Machines

William M. Stevens

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

Abstract. A simulation framework is described in which sliding tiles moving in a discrete two-dimensional grid can be put together to build machines. The tiles can perform logical and mechanical functions, and can be connected to each other. A self-replicating machine has been designed in this environment and its operation is summarised. Observations are made about the usefulness and the limitations of the machine and its environment, and several ways in which the limitations could be addressed are described. A justification of the simulation approach for modelling self-replicating systems is given.

Key words:
self-replication, self-organisation, simulation, mechatronics

1 Introduction

It has been proposed that self-replicating manufacturing systems will find applications ranging from the exploration of the galaxy [6] to the molecular-scale assembly of macroscale objects [5]. The problem of designing a system capable of constructing a range of useful objects as well as a replica of itself from a feedstock of raw materials is complex, and because of this some researchers have restricted their attention to the problem of designing self-replicating systems that work by assembling simpler subsystems. To date, only a handful of very simple physical self-replicating systems have been demonstrated (see section 4.1). All rely on a supply of pre-fabricated parts, some require rather complex parts, and some cannot construct anything other than replicas. This paper presents a simulation environment designed to assist exploration of the design space for self-replicating systems made from simple pre-fabricated parts. A specific example of a self-replicating programmable constructor within the simulation environment is given.

The possibilities that cellular automata offer for research into self-replicating systems have been widely explored. Von Neumann [15] developed a programmable constructing automaton embedded in a cellular automaton in the late 1940s and used it to prove the existence of a self-replicating automaton capable of constructing any other automaton within its domain of operation.

Since then several researchers have devised self-replicating systems embedded in cellular automata arrays. These have ranged from programmable constructors in the pattern of von Neumann's example [15, 4] to simple self-replicating loops that can do nothing but produce copies of themselves [10, 2]. Sipper gives an overview of these and other self-replicating systems [17, 18].

Research into the design and construction of physical self-replicating machines may benefit from simulation environments that offer a greater degree of physical realism than cellular automata. The simulation framework described in this paper is one such environment. In addition to devising a self-replicating structure embedded in a cellular automaton array, von Neumann also proposed what is now called his 'kinematic model' for studying the constructional capabilities of machines [15]. The system described in this paper bears some resemblance to von Neumann's proposal. The precise way in which the system described here is more physically realistic than cellular automaton environments is explained in section 4.

There are two main strands of research into self-replication using prefabricated parts. One strand is characterised by von Neumann's kinematic programmable constructor model and seeks to build program-controlled constructors capable of constructing replicas of themselves from a set of simple parts. The other strand is characterised by Penrose's replicating plywood shape system [16], in which a sequence of parts serves as a template upon which a replica sequence is built up. A more recent example of this strand of research can be found in [8]. These two strands of research may turn out to be complementary to each other. In nature we find that a template-based process underlies the replication of DNA and the construction of cellular components. Cellular components then cooperate together in a well-ordered way following a definite plan of development to construct a replica cell.

The CBlocks system is introduced in the rest of this section, and described more fully in section 2. In section 3 a self-replicating programmable constructor in the CBlocks system is presented. Section 4 places this work in the context of other work on physical self-replicating systems, and points out the advantages that it offers.

1.1 The CBlocks System

CBlocks is a system in which square tiles move and interact with each other on a two-dimensional discrete grid.

There are several different types of tile. Each type performs a specific function. There are types that perform logical functions, types that join tiles together, types that move in response to a signal and types that move other tiles. Tiles can send and receive signals to and from neighbouring tiles.

Tiles occupy one cell in the two-dimensional grid in which they exist, and can move either north, south, east or west in a single time unit. When a tile moves into a neighbouring cell that is already occupied, the occupying tile gets pushed away. There are rules that determine how signals pass between tiles,

how tiles can be connected together and how they should behave when they are connected.

Machines can be constructed from collections of tiles connected up in an appropriate way.

1.2 An Example

A simple example is presented before giving a more formal description of the system.

Fig. 1. A *not* tile.

Figure 1 shows a *not* tile with an input at the bottom and an output at the top. At the next time step, the output will be zero if the input is non-zero, or one otherwise.

Fig. 2. A *thrust* tile.

Figure 2 shows a *thrust* tile. When the input is non-zero, the tile moves down by one cell every time step

In figure 3, four successive time steps showing the behaviour of an arrangement of tiles is shown. The *thrust* tile is activated by the *not* tile connected to it. It moves one cell to the right every time step until the *not* tile on the right turns off the output of the *not* tile connected to the *thrust* tile. Note that the *thrust* tile continues moving to the right until one time step after the two *not* tiles come into contact because it takes one time step for the signal from the right-most *not* tile to propagate to the *thrust* tile.

2 A Concise Description of CBlocks

In CBlocks, time is discrete, and moves forward in steps of one unit. In one time unit a tile may move one cell to the north, south, east or west. Tiles can

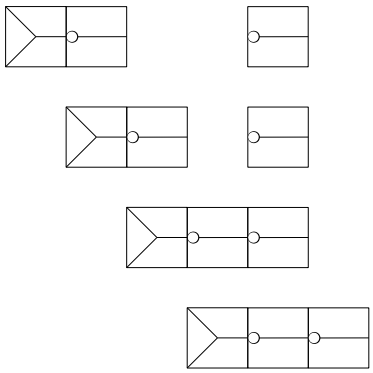

Fig. 3. A simple example.

be connected together along their edges. When two or more tiles are connected together, they move together when pushed. Rules exist to avoid conflicts that might arise when, for example, an attempt is made to push two tiles into the same cell. Since this kind of conflict does not arise in the systems described in this paper, a description of these rules is not necessary. A full discussion of the kinds of conflicts that can arise is given by Arbib in [1].

2.1 Tile Types and Signals

Two tile types have already been introduced: these were the *not* and *thrust* types. Signals were mentioned in the informal descriptions of these tiles.

The edges of tiles can be regarded as terminals through which signals can be passed between neighbouring tiles. Tiles do not need to be connected in order for signals to pass between them. Each terminal of a tile acts either as an input or as an output. If a terminal has no explicit definition, it is effectively an output producing no signal.

Signals are 32-bit integer values. The absence of a signal corresponds to a value of zero. It takes one time unit for a signal to propagate from a tile's inputs to its outputs, or for a tile to respond to signals at its inputs.

A tile's type determines how it responds to input signals, and whether it produces any output signals. A tile can be in any one of four possible orientations. The CBlocks environment is rotation symmetric, so that two structures that differ only in their orientation can be regarded as being logically and kinematically equivalent.

Table 1 describes 24 tile types, of which 23 are used in section 3. (The *RUnFuse* tile is not used, but is included in table 1 for completeness). In table 1 the letters N,S,E and W (for North, South, East and West) are used to refer to terminals and also to indicate directions. The context should indicate which usage is meant. Note that terminal labels and directions are given relative to the orientation of the tile.

1 Wire  N=S	2 Cross  N=S,E=W	3 Delta  N,E,W=S
4 Not  N=!S	5 And  N=min(E,W)	6 Or  N=max(E,W)
7 Nand  N=!(E&&W)	8 Nor  N=!(E||W)	9 Insulator 
10 Push  When S!=0, push on tile that lies N, in the N direction	12 Thrust  When S!=0, push on self in the S direction	13 RFuse  When S!=0, connect the tiles that lie N and NE
14 LFuse  When S!=0, connect the tiles that lie N and NW	15 RUnFuse  When S!=0, disconnect the tiles that lie N and NE	16 LUnFuse  When S!=0, disconnect the tiles that lie N and NW
19 RSlide  When S!=0, apply a force on tile that lies N, in the E direction	20 LSlide  When S!=0, apply a force on tile that lies N, in the W direction	21 Equal  N=(E==W)
22 Pulse  N=1 only when S changes from 0 to non-zero	24 Creator  When S is non-zero, create a tile in the N direction	25 Multiplier  N=E*W
26 Adder  N=E+W	27 Store  If S!=0 and output N==0, set output N to S. If E!=0 or W!=0, set output N to 0	32 Toggle  If S!=0, toggle the value of output N

Table 1. Tile types used for the SRM in section 3

The notation used for expressions in table 1 is that used by the C programming language, summarized in table 2.

Operator	Name and meaning
+	Plus Sum of operands
*	Times Product of operands
==	Equals 1 if operands are equal, zero otherwise
!=	Not Equals zero if operands are equal, 1 otherwise
!	Logical Not 1 if operand is zero, zero otherwise
&&	Logical And 1 if both operands are non-zero, zero otherwise
||	Logical Or 1 if any operand is non-zero, zero otherwise

Table 2. Operators used in table 1

3 A Self-Replicating Machine in CBlocks

The tile types described in the previous section have been used to make a self-replicating machine (SRM). An outline description of the machine is given in this section. There is not space in this paper to give a detailed description. Interested readers are referred to the author's website given at the top of this paper.

The SRM would be far more complex were it not for the *creator* tile type. This type allows new tiles to appear from nowhere in response to a signal.

Figure 4 illustrates the geometrical structure of the SRM, the four main parts are labelled. The instruction hopper contains a block of *store* tiles which encode a sequence that directs the SRM to move around the universe and to create tiles in such a way as to duplicate itself. This sequence of *store* tiles is referred to as the instruction tape. The instruction tape can encode instructions for building any configuration of tiles, limited only by the length of the instruction tape. To illustrate this, figure 5 shows a relatively simple configuration of tiles and table 3 gives the instruction sequence (with integer instruction codes in brackets) required to make this configuration. Notice that the orientation of tiles specified in this instruction sequence is the orientation relative to the orientation of the *creator* tile in the reader.

Self-replication is the special case where the instruction tape encodes a sequence of actions that results in a duplicate machine. In the description of the SRM that follows, the terms *parent* and *child* are used to refer to machines in a

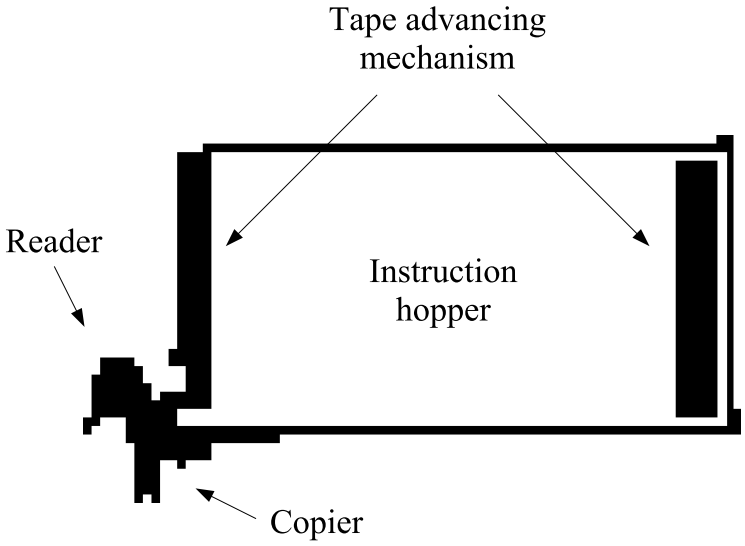

Fig. 4. The geometrical structure of the srm.

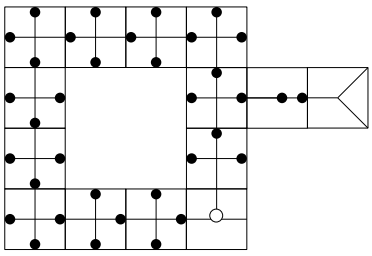

Fig. 5. An example construction.

DELTA NORTH (14)	MOVE NORTH (1004)	MOVE SOUTH (1002)
MOVE SOUTH (1002)	MOVE NORTH (1004)	ORGATE EAST (25)
DELTA WEST (15)		
MOVE SOUTH (1002)	DELTA NORTH (14)	MOVE NORTH (1004)
DELTA WEST (15)	MOVE NORTH (1004)	MOVE EAST (1001)
MOVE SOUTH (1002)	MOVE SOUTH (1002)	MOVE NORTH (1004)
DELTA WEST (15)	MOVE SOUTH (1002)	PULSER SOUTH (88)
	MOVE SOUTH (1002)	MOVE NORTH (1004)
MOVE EAST (1001)	MOVE SOUTH (1002)	MOVE SOUTH (1002)
MOVE NORTH (1004)	DELTA SOUTH (12)	MOVE EAST (1001)
MOVE NORTH (1004)		THRUSTER SOUTH (48)
MOVE NORTH (1004)	MOVE NORTH (1004)	MOVE NORTH (1004)
	MOVE EAST (1001)	MOVE SOUTH (1002)
DELTA NORTH (14)	MOVE NORTH (1004)	
MOVE NORTH (1004)	MOVE NORTH (1004)	MOVE SOUTH (1002)
MOVE SOUTH (1002)		MOVE SOUTH (1002)
MOVE SOUTH (1002)	DELTA EAST (13)	MOVE SOUTH (1002)
MOVE SOUTH (1002)	MOVE NORTH (1004)	MOVE SOUTH (1002)
MOVE SOUTH (1002)	MOVE SOUTH (1002)	
DELTA SOUTH (12)	MOVE SOUTH (1002)	MOVE WEST (1003)
	DELTA EAST (13)	MOVE WEST (1003)
MOVE NORTH (1004)	MOVE SOUTH (1002)	
MOVE EAST (1001)	DELTA EAST (13)	

Table 3. Instructions required for constructing figure 5

relationship where one instance of the machine has constructed or is constructing another.

Figure 6 shows how the instruction tape is arranged in the machine. The arrows show the direction in which *store* tiles move as the tape is advanced. The tape-advancing mechanism ensures that the tape advances one tile at a time and that the arrangement shown in Figure 6 is maintained.

The *reader* contains logic that interprets signals from the instruction tape and acts upon them. Figure 7 shows the logical structure of the reader.

The SRM uses a *creator* tile to create new tiles as they are needed. This tile creates a new tile whose type and orientation (i.e. orientation relative to the *creator* tile in the reader) are dependant on the value of the input signal that it receives. A signal value of $4T + D$ encodes a tile of type T and orientation D. Values of T for each tile type are given in table 1. The mapping between values of D and possible orientations is $\{(0, N), (1, E), (2, S), (3, W)\}$.

Fig. 6. The arrangement of the instruction tape.

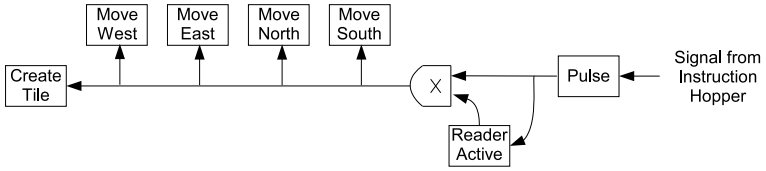

Fig. 7. The logical structure of the reader.

Some of the *store* tiles in the instruction tape encode values which tell the SRM to perform an action. The values used are as follows:

1001 = move east
1002 = move south
1003 = move west
1004 = move north
1005 = switch between *read* and *copy* phases
1006 = do nothing

No explicit instruction is needed in order to tell the reader to fuse newly created tiles together, since the reader contains *fuser* tiles that are always active and which fuse together any tiles that pass in front of them.

The *copier* is responsible for creating a duplicate instruction tape in a child SRM. Figure 8 shows the logical structure of the copier.

The replication cycle has two phases: the reading phase and the copying phase. During the reading phase the instruction tape is interpreted by the reader. The last instruction in the instruction sequence (code 1005) causes the machine to toggle between the reading and copying phases. During the copying phase, the parent sends signals to the child which cause a copy of the instruction tape to be created in the child SRM. The child machine is then complete and can begin constructing its own child. The parent machine switches back to the reading

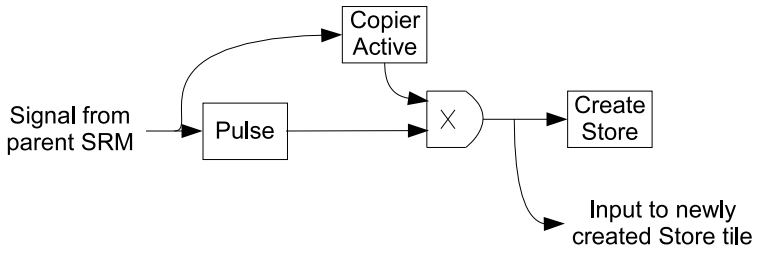

Fig. 8. The logical structure of the copier.

phase and the replication cycle begins again. Figures 9 and 10 show two snapshots of the SRM in action, showing which phase it is in and what it is doing at each snapshot.

Figure 11 shows a parent SRM and the child SRM that it has produced. Notice that the child is constructed so as to be oriented 90 degrees anticlockwise with respect to the parent. This is done so that successive generations of SRMs will fill up the two-dimensional universe. It might be argued that because of this difference in orientation the child is not an exact replica of the parent. However, since the CBlocks environment is rotationally symmetric the logical and kinematical behaviour of parent and child can reasonably be regarded as equivalent.

Figure 12 shows the state of the universe after the initial SRM has produced two child SRMs, the first of which has produced a child of its own.

3.1 Part counts

The SRM is made from 2311 tiles, including the 1777 *store* tiles in the instruction tape. 23 different types of tile are used (see table 1). The number of each type used is given in table 4.

4 CBlocks and Physical Self-Replicating Machines

In the introduction it was asserted that the CBlocks environment is more physically realistic than cellular automaton environments. This statement needs justification. Cellular automaton environments can of course be made from arrays of discrete parts, with each physical part corresponding directly to a cell in the abstract environment. A self-replicating system in such an environment would be able to alter the internal state of a discrete part in the array, but this would be the limit of its effect on the physical environment. Such a system does not harness the mechanics of the physical environment for the purpose of self-replication.

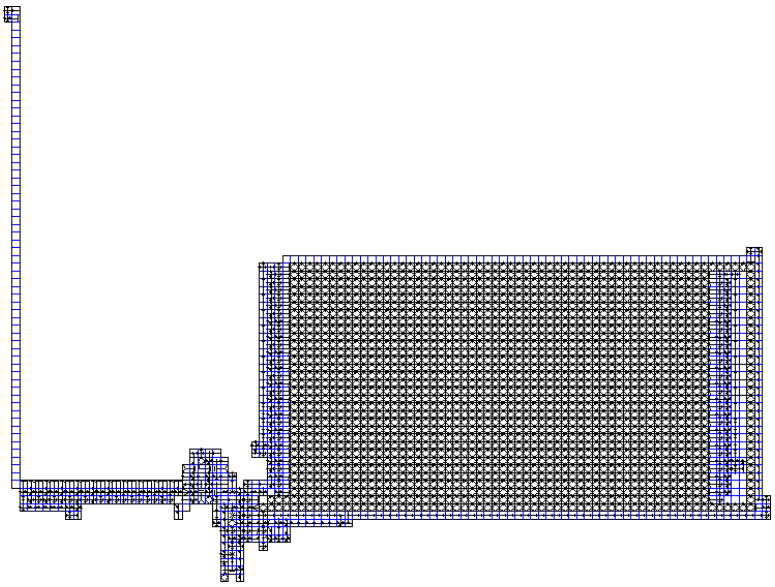

Fig. 9. The parent SRM is in the reading phase and is part way through constructing a child SRM.

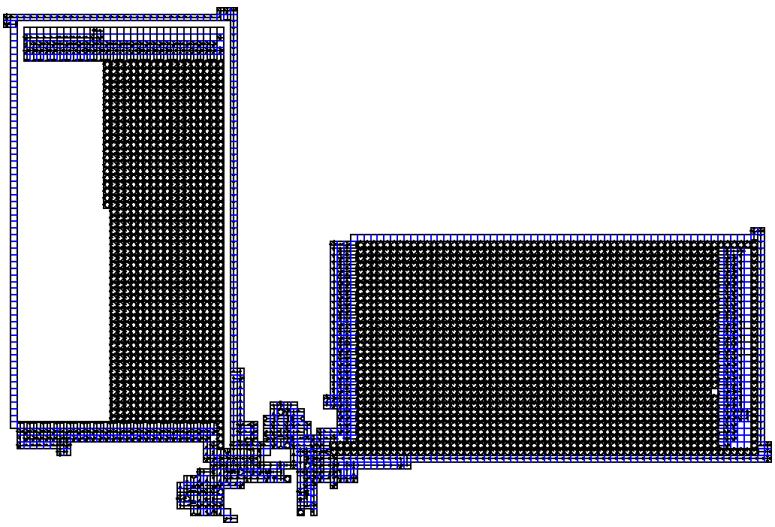

Fig. 10. The parent SRM has finished the reading phase and is part way through the copying phase.

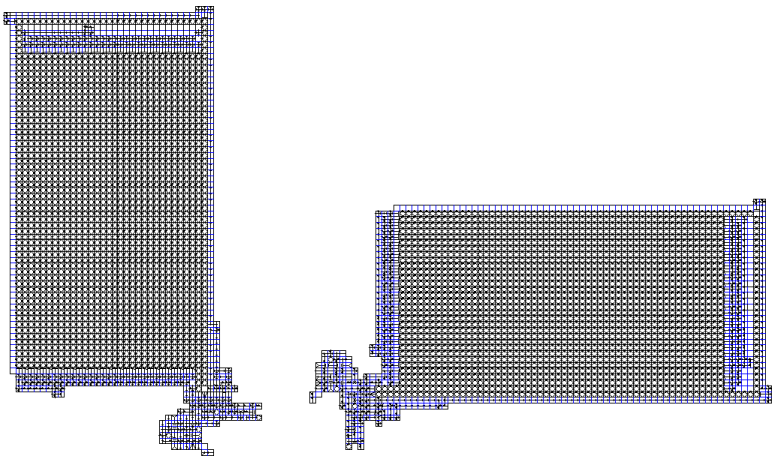

Fig. 11. A parent SRM has produced a child

Store	1777+5
Delta	172
Wire	124
Insulator	108
Pusher	32
Left-slider	18
Right-slider	15
Toggle	12
15 Others	48 (less than 6 of each, four tile types only used once)

Table 4. Count of the types of tile used in the SRM.

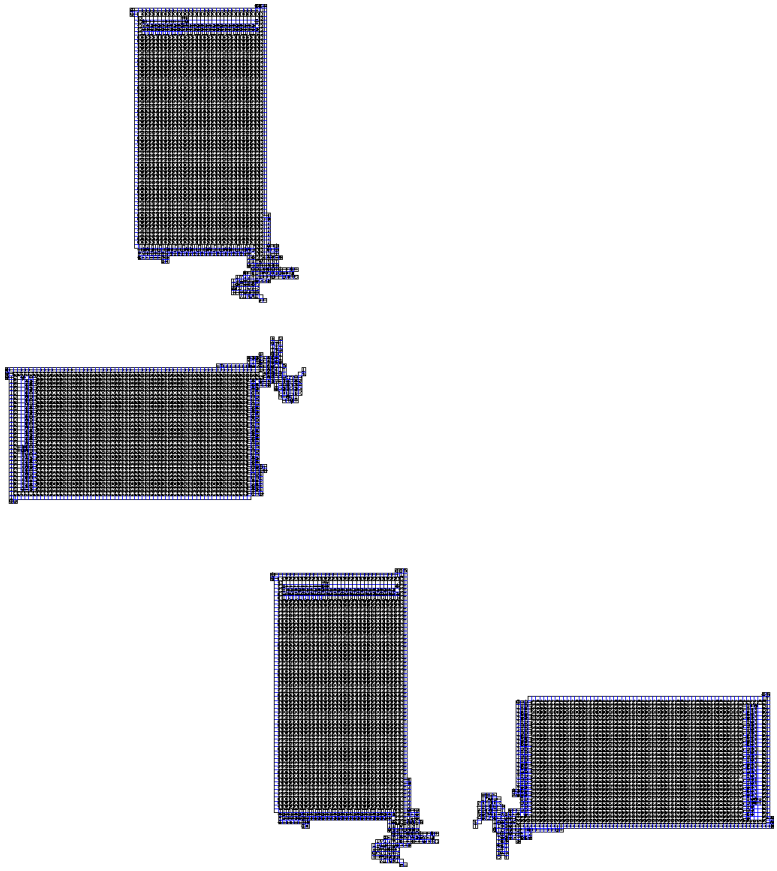

Fig. 12. SRMs after two generations

A physical self-replicating system capable of building itself from component parts would have to make use of the mechanics of the environment in which the component parts function in order to replicate itself. The self-replicating system described in this paper does this within the CBlocks environment. This environment has rules of motion and interaction loosely based upon the laws of motion and the mechanical interactions of physical machines. In this sense, the CBlocks environment can be said to be more physically realistic than cellular automaton environments.

Matter is not conserved in the CBlocks environment, since a *creator* tile can create other tiles from nowhere. At first sight it seems that a *creator* tile is not at all physically realistic. However, it is possible to envisage physical systems in which something like a *creator* tile can be made. For example, if a physical model based on CBlocks existed on a two-dimensional surface and a second surface were placed just above this surface, the second surface could contain a disorganised collection of tiles, moving about at random. A *creator* tile on the lower surface needing to create a tile could wait for a tile of the correct type and in the correct orientation to pass above it on the upper surface, and then cause it to be transferred from the upper to the lower surface.

It may be possible to devise an SRM in CBlocks which is not dependant on a *creator* tile. This SRM could fetch the tiles needed to build a copy of itself from a known location, or alternatively it could forage for tiles in a disorganised collection.

4.1 Other Work on Physical Self-Replicating Machines

Other work related to physical self-replication is summarised below in chronological order.

Penrose devised a set of plywood shapes that could be placed into a container and agitated. The shapes would remain in a disordered state, unless seeded by a particular configuration of two shapes, in which case other shapes would tend to pair up and adopt the same configuration [16].

A NASA summer study in 1980 investigated the possibility of building a self-replicating factory on the moon [6].

More recently, Chirikjian et al devised LEGO robots that could put together other robots from three complex parts [3].

Moses developed a set of plastic blocks that could be used to make a three-dimensional constructor, capable of building another constructor under the control of a computer or a human operator [13].

Zykov et al. built a system in which robots made from identical subunits containing power sources, motors and processors could build other robots from the same subunits, including copies of themselves [20].

Malone and Lipson developed a compact free-form fabrication system capable of making various components including batteries, wires and flexible joints [11].

Freitas and Merkle published a comprehensive work which describes proposals for and implementations of physical self-replication, along with some of the engineering issues surrounding physical self-replication [7].

Griffith et al. built a system for investigating physical template-based self-replication. This system consists of a programmable unit which can communicate with and connect to neighbouring units. Behaviour required for self-replication can be programmed into a large number of units, which will form replicas when seeded with a template structure [8].

The works described above can be categorised as follows: [16], [3] and [8] are systems made from a small number of parts, designed for the purpose of self-replication but with little ability to do anything else. [13] and [20] are systems made from a larger number of parts, taken from a small set of pre-fabricated part types. These systems are program-controlled and are therefore able to construct a wide range of machines made from the pre-fabricated parts. [6] and [11] are systems which make their own parts from raw material feedstock. These systems offer a great deal of flexibility: not only can the arrangement of parts be specified by a program, the structure of the parts themselves can also be specified by a program. [6] is a long way from being realised. [11] can make a small number of component parts and is some way from being able to make the same set of parts from which it is made and then assemble those parts into a replica.

4.2 The Value of a Simulation Approach

Within this categorisation scheme, the systems described in [13] and [20] are the most similar to the programmable SRM described in this paper so it is worth examining these in more detail to see what advantages a simulation approach offers to this class of system.

The plastic blocks in Moses's system [13] are of 11 different types and are designed to permit the construction of a constructing machine based around a controllable manipulator that can pick up blocks one at a time, position them in three dimensions and then slot them into a structure being built. Moses's system is controlled by an external program: blocks containing motors are fed signals from a computer or from a human operator that lies outside the environment.

The system of Zykov et al. [20]. is based around a single type of block that combines processing, connective and motor functions into a single unit. Four of these units can be put together to make a machine capable of manipulating other units and arranging them into a replica configuration. In contrast to Moses's approach of using a set of simple parts to build a more complex structure under external control, Zykov's system is made from complex parts, each of which is capable of containing a description of the steps required to replicate the whole system.

Both systems can be regarded as steps towards the goal of making a fully autonomous self-replicating system with a low component-part complexity and high constructional capability, which will be referred to as goal G. [13] has a low part complexity and a constructional capability limited only by the 11 part types available and the size of the domain that the constructor operates in, but is not fully autonomous. [20] is fully autonomous and also has a wide constructional capability, but has a very high part complexity.

Building physical prototypes is expensive and time-consuming. To make headway towards goal G starting from a system like [13] it would be necessary to design a control unit from the set of available part types, perhaps augmented with several more to facilitate the processing of digital information. Simulation would greatly help the design process. [13] has a deliberately restricted set of possible mechanical interactions between parts, and could therefore be simulated by a CBlocks type model, which simulates only logical, geometrical and kinematical interactions between parts and between subsystems. By omitting detailed simulation of mechanical interactions from a simulation model, simulation time is greatly reduced.

The designers of system [20] had both simulation and physical construction in mind when designing their system. A simulation model that models the geometrical and logical constraints of the system was used to come up with manually designed and automatically evolved self-replicating structures [14]. In the physical prototype, a microcontroller was used to implement the controlling logic for the parts, and the logical communication that takes place between parts is complex.

One way to progress towards G starting from [20] would be to separate out the different functions of the single part used in this system into perhaps three or four different types of part. One part for logical processing, one for movement, and one or two for connecting/disconnecting other parts. An attractive feature of the system in [20] is that the swiveling half-cube method for moving parts around can be used both for translation and rotation of parts.

Implementing a controller for such a system derived from [20] using logical processing parts containing a simple logical element such as a boolean logic gate would be challenging without the aid of a simulation model.

4.3 Computation and Construction

Several researchers have attempted to establish criteria that can be used to distinguish between trivial self-replicating systems such as crystals growing in solution and fire in a flammable medium and non-trivial systems such as living cells and von Neumanns self-replicating automaton.

Starting with Burks [15], some researchers have used the capability for universal computation as the sole distinguishing criterion. Both Herman [9] and Langton [10] criticise Burks on different grounds. Herman presents a self-replicating automaton capable of universal computation that seems intuitively trivial, and Langton argues that the simplest living cells are not capable of universal computation and yet seem intuitively non-trivial.

McMullin [12] gives a detailed critique of Burks criterion and clears up some of the confusion surrounding the issue of trivial versus non-trivial self-replication by pointing out that von Neumann's self-replicating automaton was not designed as an end in itself, but was part of the answer to a question that von Neumann posed about the ability of a machine to create other machines more complex than itself. Therefore, von Neumann was concerned about the class of objects that his automaton could be programmed to construct.

Researchers interested in the applications of physical self-replication share the same concern, and in addition are concerned with the complexity of the component parts of a machine, and the complexity of a machine relative to the complexity of its component parts.

Logical universality and the capability for universal computation enter this scene almost incidentally for the following reasons: Firstly a replicator that is controlled by a universal computer is likely to have a larger constructional capability than one that is not. Secondly a replicator may not itself contain a universal computer, but may be capable of constructing one (the SRM described in this paper is an example of this).

At the present time, a machine's constructional capability is not so easy to quantify and hold up for comparison with other machines as its computational capability, since constructional capability depends on the environment that the machine operates in. Computational capability depends only on the logical structure of a system, where as constructional capability can depend on the logical, kinematical, mechanical and physical structure of a system (i.e. how flexibly a system can be programmed, the range of its movements, how its parts interact with each other mechanically, and what materials its parts are made from).

5 Conclusion

A simulation environment has been developed in which a self-replicating machine has been constructed.

The SRM is a moving programmable constructor made from 23 different types of part. It is controlled by a looping sequence of instructions contained within the body of the machine. The SRM uses a special *creator* part that can create other parts out of nowhere.

There is some redundancy in the set of parts used by the machine. For example, there are 4 types of tile which exert forces, 12 types of tile which perform arithmetic or combinational logic operations and 3 types of tile that connect or disconnect other tiles. In choosing the set of tiles used in this paper, a trade-off had to be made between SRM size (and simulation time requirements) on the one hand, and tile complexity/redundancy on the other.

The system could be extended to three dimensions, with the advantage that in three dimensions it is possible to access unit parts (cubes) from six directions instead of four, and the routing of signals around machines becomes easier. From the perspective of physical implementation, a three dimensional model poses some problems. How should parts that are not in contact with the ground be supported? How can power be routed to parts that are surrounded on all sides by other parts?

Future work is expected to result in an SRM that uses a greatly reduced part set, with only one type of tile for each different class of function. The need for a *creator* tile will be removed by having the machine forage for parts in a disorganised collection and testing each part that is found to determine its type.

CBlocks was developed with the aim of creating an environment with a greater degree of physical realism than cellular automata environments and in which an SRM made from simple component parts could be constructed. This aim has been met. An environment called Nodes which uses Newtonian laws of motion was developed side-by-side with CBlocks in order to explore this aim further. This is described in [19].

6 Obtaining CBlocks

Software and C++ source code for the CBlocks environment are available at the following the URL:

http://www.srm.org.uk

The website also contains files needed to simulate the SRM described in this paper and a more detailed description of the structure of the SRM.

References

1. Arbib, M.A.: Theories of Abstract Automata. Prentice-Hall, Englewood Cliffs, New Jersey 355–361 (1969)
2. Byl, J.: Self-Reproduction in Small Cellular Automata. Physica D **34** 295–299 (1989)
3. Chirikjian, G.S., Zhou, Y., Suthakorn, J.: Self-replicating Robots for Lunar Development. IEEE/ASME Transactions on Mechatronics **7(4)** 462–472 (2002)
4. Codd, E.F.: Cellular Automata. Academic Press, New York (1968)
5. Drexler, K.E.: Engines of Creation: The Coming Era of Nanotechnology Anchor Press/Doubleday, New York (1986)
   http://www.foresight.org/EOC/
   Cited on 25 November 2006
6. Freitas, R.A. Jr.: Report on the NASA/ASEE summer study on advanced automation for space missions. Journal of the British Interplanetary Society **34** 407–408 (1981)
7. Freitas, R.A. Jr., Merkle, R.C.: Kinematic Self-Replicating Machines. Landes Bioscience, Georgetown Texas (2004)
   http://www.molecularassembler.com/KSRM.htm
   Cited on 25 November 2006
8. Griffith, S., Goldwater, D. Jacobson, J.M.: Robotics: Self-replication from random parts. Nature **437** 636 (2005)
9. Herman, G.T.: On Universal Computer Constructors. Information Processing Letters **2** 61–64 (1973)
10. Langton, C.G.: Self-reproduction in cellular automata. Physica D **10** 135–144 (1984)
11. Malone, E., Lipson, H.: Functional Freeform Fabrication for Physical Artificial Life. Proc. 9th International Conference on the Simulation and Synthesis of Living Systems. MIT Press, Boston Massachusetts 100–105 (2004)

12. McMullin, B.: John von Neumann and the Evolutionary Growth of Complexity: Looking Backwards, Looking Forwards... Artificial Life VII: Proceedings of the Seventh International Conference. MIT Press, Boston Massachusetts 467–476 (2000)
http://www.eeng.dcu.ie/ alife/bmcm-2000-01/
Cited on 25 November 2006
13. Moses, M.: A Physical Prototype of a Self-Replicating Universal Constructor. Masters Thesis, Department of Mechanical Engineering, University of New Mexico (2001).
http://www.home.earthlink.net/˜mmoses152/SelfRep.doc
Cited on 25 November 2006
14. Mytilinaios, E., Desnoyer, M., Marcus, D., Lipson, H.: Designed and Evolved Blueprints For Physical Self-Replicating Machines. Proc. 9th International Conference on the Simulation and Synthesis of Living Systems. MIT Press, Boston Massachusetts 15–20 (2004)
15. Von Neumann, F.: Theory of Self-Reproducing Automata. Edited and completed by A.W. Burks. University of Illinois Press, Urbana Illinois 81–82 (1966)
16. Penrose, L.S.: Self-reproducing machines. Scientific American **200(6)** 105–114 (1959)
17. Sipper, M.: Fifty years of research on self-replication: An overview. Artificial Life **4(3)** 237–257 (1998)
18. Sipper, M.: The Artificial Self-Replication Page (1998-Present).
http://www.cs.bgu.ac.il/˜sipper/selfrep
Cited on 25 November 2006
19. Stevens, W.M.: Nodes: An Environment for Simulating Kinematic Self-Replicating Machines. Proc. 9th International Conference on the Simulation and Synthesis of Living Systems. MIT Press, Boston Massachusetts 39–44 (2004)
http://www.srm.org.uk/papers/nodespaper.pdf
Cited on 25 November 2006
20. Zykov, V., Mytilinaios, E., Adams, B., Lipson, H.: Self-Reproducing Machines. Nature **435** 163–164 (2005)