



The United Nations
University

UNU-IIST

International Institute for
Software Technology

Refining emergent properties

J. W. Sanders and Graeme Smith

July 2009

UNU-IIST and UNU-IIST Reports

UNU-IIST (United Nations University International Institute for Software Technology) is a Research and Training Centre of the United Nations University (UNU). It is based in Macao, and was founded in 1991. It started operations in July 1992. UNU-IIST is jointly funded by the government of Macao and the governments of the People's Republic of China and Portugal through a contribution to the UNU Endowment Fund. As well as providing two-thirds of the endowment fund, the Macao authorities also supply UNU-IIST with its office premises and furniture and subsidise fellow accommodation.

The mission of UNU-IIST is to assist developing countries in the application and development of software technology.

UNU-IIST contributes through its programmatic activities:

1. Advanced development projects, in which software techniques supported by tools are applied,
2. Research projects, in which new techniques for software development are investigated,
3. Curriculum development projects, in which courses of software technology for universities in developing countries are developed,
4. University development projects, which complement the curriculum development projects by aiming to strengthen all aspects of computer science teaching in universities in developing countries,
5. Schools and Courses, which typically teach advanced software development techniques,
6. Events, in which conferences and workshops are organised or supported by UNU-IIST, and
7. Dissemination, in which UNU-IIST regularly distributes to developing countries information on international progress of software technology.

Fellows, who are young scientists and engineers from developing countries, are invited to actively participate in all these projects. By doing the projects they are trained.

At present, the technical focus of UNU-IIST is on formal methods for software development. UNU-IIST is an internationally recognised center in the area of formal methods. However, no software technique is universally applicable. We are prepared to choose complementary techniques for our projects, if necessary.

UNU-IIST produces a report series. Reports are either Research R, Technical T, Compendia C or Administrative A. They are records of UNU-IIST activities and research and development achievements. Many of the reports are also published in conference proceedings and journals.

Please write to UNU-IIST at P.O. Box 3058, Macao or visit UNU-IIST's home page: <http://www.iist.unu.edu>, if you would like to know more about UNU-IIST and its report series.

G. M. Reed, Director



The United Nations
University

UNU-IIST

International Institute for
Software Technology

P.O. Box 3058
Macao

Refining emergent properties

J. W. Sanders and Graeme Smith

Abstract

Systems which exhibit *emergent* behaviour, *i.e.*, behaviour not determined by the behaviours of the constituents when considered in isolation, are becoming more common due to increasing use of distributed and decentralised designs. There have been claims that formal methods, and particularly refinement, can not be used to derive systems with emergent behaviour. In this paper, however, we argue that they can. To prove the point, we perform a refinement of an oft-cited example of emergence: the ‘glider’ pattern from Conway’s Game of Life.

Jeff Sanders is Principal Research Fellow at UNU-IIST with interests largely in Formal Methods. He acknowledges financial support from the ARC Centre for Complex Systems (ACCS), Australia, and the Macao Science and Technology Development Fund under the PEARL project, grant number 041/2007/A3.

Graeme Smith is a senior lecturer in the School of Information Technology and Electrical Engineering at the University of Queensland, Australia. He is well known for his work on Formal Methods, Z and object orientation.

Contents

1	Introduction	1
2	One-dimensional Game of Life	2
2.1	Specification	2
2.2	Implementation	2
2.3	Refinement	3
2.4	Variation 1	4
2.5	Variation 2	5
3	Two dimensions	5
3.1	The rules	6
3.2	Helpful geometric results	8
3.3	Applying geometry	10
3.4	Headings	12
3.5	Two-dimensional gliders	13
3.6	Refinement	14
4	Conclusion	16

1 Introduction

The ongoing decrease in size and cost of microprocessors and storage devices is leading to the development of increasingly distributed and decentralised systems. Components of such systems have limited access or often no access to global information and must operate on local information gained via interaction with neighbouring components. Often these systems exhibit *emergent* behaviour: global behaviour that is not determined by the behaviours of the constituents when considered in isolation.

Reluctance to adopt formal methods when engineering such systems has arisen partly from their similarities with *complex systems*. The research on complex systems, however, has focussed on the modelling of *existing* systems (both natural and man-made, *e.g.*, the Internet), and the prediction of their global properties. When we model an existing system, unknown discontinuities in behaviour may not be modelled, and hence proof techniques may not be successful in uncovering emergent behaviour. Furthermore, it is claimed that some complex systems exhibit *strong emergence* [1] (*e.g.*, the mind) and therefore, by definition, proofs of how their behaviour arises cannot be constructed.

When we engineer *new* systems, however, we are not trying to prove the existence of emergent behaviours. Rather we start with the emergent behaviour we require (which may include the avoidance of undesirable behaviours), and develop a design which gives rise to it. To provide assurance that a design is sufficient, the emergent behaviour must be a consequence of the component interactions within the design. Hence, we are interested only in systems where that is the case, *i.e.*, systems which exhibit *weak emergence* [1]. Note that many classic examples of emergence from the field of Complex Systems, such as ant foraging and bird flocking behaviours, are examples of weak emergence.

So are standard formal methods, and in particular refinement, applicable to the engineering of systems with (weak) emergent properties? Polack and Stepney [9] argue that they are not. They posit the emergent behaviour of the ‘glider’ pattern of Conway’s Game of Life¹ [6, 2] as an example where refinement cannot be used.

Conway’s Game of Life is a cellular automaton which simulates the evolution of a grid of cells. The cells evolve according to the following four rules.

1. A live cell with less than two live neighbours dies (of isolation).
2. A live cell with more than three live neighbours dies (of overcrowding).
3. A live cell with two or three live neighbours remains a live cell.
4. A dead cell with exactly three live neighbours becomes a live cell.

A live cell is represented by a cell which is on and a dead cell by one that is off. The grid of cells is infinite, although this is usually represented in implementations using a finite grid arranged as a torus, or by simply having the finite grid surrounded by cells which cannot turn on.

¹An executable version can be found at <http://www.bitstorm.org/gameoflife/>.

Many different patterns can be formed in the Game of Life including dynamic patterns such as the glider which translates itself across the grid. Polack and Stepney argue that an abstract specification of the movement of a glider cannot be refined to the rules of the Game of Life shown above. Their justification is that when an implementation exhibits emergence, the specification and implementation (and even the languages in which they are expressed) must be too disparate.

In this paper, however, we show how the abstract behaviour of the glider may be specified and subsequently refined to an array of cells following the rules of the Game of Life using the simplest possible standard refinement techniques. Our approach builds on ideas proposed in [8], elaborated in [11] and exemplified in [12].

To provide a simple illustration of our approach, we begin in Section 2 with the specification and refinement of dynamic patterns, including a glider, in a one-dimensional Game of Life. To show that in making that simplification we have not inadvertently brought together the two levels of abstraction, in Section 3 we provide a specification and refinement of the glider in the full two-dimensional Game of Life. The approach is the same, only the complexity of the detail is increased. We conclude with a discussion of related work relating to the question of refining emergent properties in Section 4.

2 One-dimensional Game of Life

2.1 Specification

To describe one-dimensional patterns we consider a system whose state is a Boolean-valued function on \mathbb{Z} . Thus at any (discrete) time $t : \mathbb{N}$ the state at $n : \mathbb{Z}$ is either *true* or *false*; we write it

$$x[n, t] : \mathbb{B}.$$

By abuse of notation we think of a location $n : \mathbb{Z}$ whose state is *true* as being ‘occupied’ and one whose state is *false* as being ‘vacant’. In preparation for the implementation to come, a location is also called a ‘cell’. Note however that implementations that do not use cells are also possible: the location is just a discrete point on some plane (the minimum information needed to describe movement formally).

A *glider* is thought of informally as a ‘cell moving right from the origin, one location per time step and starting from the origin at time 0 (but with all other locations vacant)’. It is specified:

$$\text{glider} := \forall n : \mathbb{Z}, t : \mathbb{N} \cdot x[n, t] = (n = t).$$

2.2 Implementation

The implementation we are interested in is composed of cells, one for each integer. However since the cells are updated synchronously, we consider state to be a function, $x : \mathbb{Z} \rightarrow \mathbb{B}$, whose value at the next time step is $x' : \mathbb{Z} \rightarrow \mathbb{B}$. The state of a cell before update is $x[n]$ and afterwards is $x'[n]$.

We impose a constraint of locality: each cell's state at the next time step, $x'[n]$, is defined in terms of the current states of itself $x[n]$ and its immediate neighbours $x[n-1]$ and $x[n+1]$.

A simple choice of design, conveniently expressed by (a deterministic) transition predicate, is

$$x'[n] = x[n-1] \wedge \neg x[n] \wedge \neg x[n+1].$$

Thus a cell 'appears to move one location to the right' if in the current state it is occupied but the next two cells to its right are vacant. There are many alternatives to this particular equation, keeping the specification in mind. Such a design phase is expected at each step of the incremental method and this particular step ensures that the specification is met by a straightforward design.

The implementation has the initial state

$$init := \forall n : \mathbb{N} \cdot x[n] = (n = 0)$$

with only the cell at the origin occupied. Thereafter, at each time step all cells are updated simultaneously according to their transition equation. We express this using the following naive notation (in order to remain independent of a formalism—but it can be expressed simply in any of the usual formalisms) for the resulting one-dimensional cellular automaton

$$ca1 := init \ ; \ \mathbf{do} \ \mathit{true} \ \rightarrow \forall n : \mathbb{Z} \cdot x[n] := x'[n] \ \mathbf{od}.$$

2.3 Refinement

The required refinement is

$$glider \sqsubseteq ca1$$

but in fact we can show equality in this simple (deterministic) case by a straightforward induction over time as follows.

For time $t = 0$ we have

$$init = glider[0/t].$$

Assuming that the glider condition holds at some time $t : \mathbb{N}$

$$(1) \quad \forall m : \mathbb{Z} \cdot x[m, t] = (m = t)$$

we wish to infer that it also holds at time $t + 1$. Now if $n : \mathbb{Z}$ and $t : \mathbb{N}$ are arbitrary then

$$x[n, t + 1]$$

$$\begin{aligned}
&= && \text{the meaning of the implementation} \\
&x'[n] \\
&= && \text{induction hypothesis (1)} \\
&\forall m : \mathbb{Z} \cdot x[m] = (m = t) \\
&\wedge \\
&x'[n] \\
&= && \text{definition of transition equation} \\
&\forall m : \mathbb{Z} \cdot x[m] = (m = t) \\
&\wedge \\
&(x[n-1] \wedge \neg x[n] \wedge \neg x[n+1]) \\
&= && \text{calculus} \\
&(n-1 = t) \wedge (n \neq t) \wedge (n+1 \neq t) \\
&= && \text{calculus} \\
&(n = t+1).
\end{aligned}$$

Generalising over $n : \mathbb{Z}$ we infer (1) with t replaced by $t+1$ as required.

Although it embodies the principle being made here —that the implementation is a refinement of the specification— this design is simple because it is strongly influenced by the desired invariant (which may be thought of as emergent).

We now consider two slightly more complex alternatives.

2.4 Variation 1

A ‘floater’ is a pattern in which, at time step $t : \mathbb{N}$, cell $x[t]$ is occupied and all cells to its right are vacant; but the cells to the left of $x[t]$ are unconstrained:

$$\text{floater} := \forall n, t : \mathbb{N} \cdot (n \geq t) \Rightarrow x[n, t] = (n = t).$$

The implementation is the cellular automaton with the same initial condition as before, but transition predicate

$$x'[n] = x[n] \neq x[n-1]$$

and the proof of correctness is no more complicated than before, in view of the implication in the specification.

It is also possible to prove further emergent properties of this implementation. For example, it can be shown that after the first time step, $x[0]$ is always vacant, $x[1]$ is always occupied, but that $x[2]$ alternates

between being vacant and occupied:

$$\forall t : \mathbb{N}_1 \cdot \neg x[0] \wedge x[1] \wedge x[2] = \text{even}(t).$$

With the usual initialisation, the following alternative implementation can also be shown to implement a floater:

$$x'[n] = \text{odd}(\#\{x[n-1], x[n], x[n+1]\}).$$

In this case, the emergent properties are quite different. From initialisation, $x[0]$ is always occupied, and $x[t-1]$ alternates between being vacant and occupied:

$$\forall t : \mathbb{N} \cdot x[0] \wedge x[t-1] = \text{odd}(t).$$

2.5 Variation 2

A *k-glider*, for $k : \mathbb{N}$, is thought of as ‘a cell moving right one location per time step starting from the origin at time k , whose immediate neighbours are always vacant but with other locations unconstrained’. In other words:

$$\begin{aligned} k\text{-glider} &:= \forall n : \mathbb{Z}, t : \mathbb{N} \cdot \\ &\quad t-k-1 \leq n \leq t-k+1 \Rightarrow x[n, t] = (n = t-k \wedge n \geq 0). \end{aligned}$$

Evidently a simple glider is a 0-glider.

A ‘glider gun’ is a state which periodically produces gliders. It can be specified:

$$\exists k : \mathbb{N} \cdot \forall t : \mathbb{N} \cdot tk\text{-glider}.$$

This specification is nondeterministic in the period k with which gliders are produced. Note that for $k = 2$ it is infeasible (not enough vacant cells!) but for $k \geq 3$ it is feasible.

Choosing the simplest implementation, we set

$$\begin{aligned} x'[n] &= x[n-1] \wedge \neg x[n] \wedge \neg x[n+1] \\ &\quad \triangleleft n \neq 0 \triangleright \\ &\quad \neg x[n-1] \wedge \neg x[n] \wedge \neg x[n+1]. \end{aligned}$$

The condition which determines when $x[0]$ is on simplifies to $\neg x[-1] \wedge \neg x[0] \wedge \neg x[1]$ and hence is satisfied when each glider reaches location 2. Therefore, every third time step a glider is produced.

3 Two dimensions

In this section we consider the integer plane \mathbb{Z}^2 and the standard rules for the Game of Life, as given in the Introduction. The state of Game of Life consists again of a function, this time $x : \mathbb{Z}^2 \rightarrow \mathbb{Z}^2$, whose state pointwise (or cell-wise) is written $x[m, n]$ and whose state pointwise after a transition is written $x'[m, n]$.

label	$x[m,n]$	$v(m,n)$	$x'[m,n]$
a		$\neq 2,3$	<i>false</i>
b	<i>true</i>	2	<i>true</i>
c	<i>true</i>	3	<i>true</i>
\bar{b}	<i>false</i>	2	<i>false</i>
\bar{c}	<i>false</i>	3	<i>true</i>

Figure 1: Labels for four important types of cell in the Game of Life.

3.1 The rules

The neighbourhood of a cell $(m,n) : \mathbb{Z}^2$ consists of (m,n) and its eight adjacent cells

$$N(m,n) := \{(i,j) : \mathbb{Z}^2 \mid |m-i| < 2 \wedge |n-j| < 2\}.$$

The number of occupied neighbours of a cell (m,n) consists of the number of occupied cells in $N(m,n)$ not including (m,n) itself

$$v(m,n) := \#\{(i,j) \in N(m,n) \mid (i,j) \neq (m,n) \wedge x[i,j]\}.$$

Evidently $0 \leq v(m,n) \leq 8$.

The state of the Game of Life is a function from locations to Booleans, $x : \mathbb{Z}^2 \rightarrow \mathbb{B}$, and its transition rules simplify to this:

$$(2) \quad x'[m,n] := \begin{array}{l} v(m,n) = 3 \\ \vee \\ v(m,n) = 2 \wedge x[m,n]. \end{array}$$

For example, if a cell has at most one occupied neighbour then its next state is unoccupied, regardless of its current state; similarly, if a cell has at least 4 occupied neighbours. Thus

$$v(m,n) \neq 2,3 \Rightarrow \neg x'[m,n].$$

A cell $x[m,n]$ satisfying $v(m,n) \neq 2,3$ we say is of *type a*. For the analysis of particular configurations, it is convenient to document the remaining cases by introducing four further types of cell. Their definition is given in Figure 1 and an example appears in Figure 2.

We wish to think, as in one dimension, in terms of the movement of shapes with time. If $A \subseteq \mathbb{Z}^2$ is the set of all occupied cells then σA consists of all cells that are occupied after a transition:

$$(3) \quad \sigma A := \{(m,n) \mid x'[m,n]\}.$$

a	a	a	a	a	a	a
a	a	a	\bar{b}	a	a	a
a	a	a	a	b	\bar{b}	a
a	a	\bar{c}	a	c	\bar{b}	a
a	a	\bar{b}	b	\bar{c}	a	a
a	a	a	a	a	a	a
a	a	a	a	a	a	a

Figure 2: An initial configuration of the two-dimensional glider with cells labelled using the convention of Figure 1.

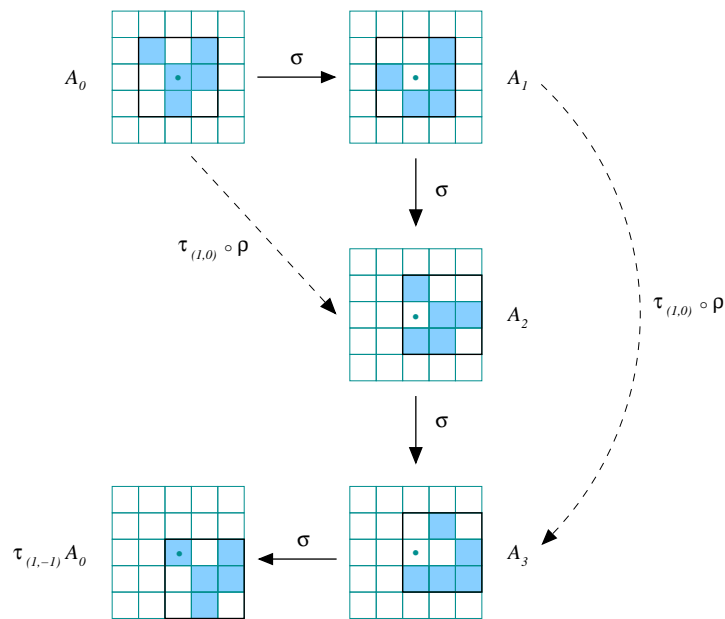


Figure 3: Relationships between glider configurations. The origin, marked as a dot, lies at the centre of each 5×5 array.

The iterate σ^k of the function σ gives the cells that are occupied after k transitions.

For example we shall see later that with these sets, depicted in Figure 3,

$$(4) \quad A_0 := \{(-1, 1), (0, 0), (0, -1), (1, 0), (1, 1)\}$$

$$(5) \quad A_1 := \{(-1, 0), (0, -1), (1, -1), (1, 0), (1, 1)\}$$

$$(6) \quad A_2 := \{(0, 1), (0, -1), (1, 0), (1, -1), (2, 0)\}$$

$$(7) \quad A_3 := \{(0, -1), (1, 1), (1, -1), (2, 0), (2, -1)\}$$

we have

$$(8) \quad \sigma A_0 = A_1$$

$$(9) \quad \sigma A_1 = A_2$$

$$(10) \quad \sigma A_2 = A_3.$$

Notes. (a) We have avoided use of the notation A' which, if used for the pointwise lifting of $x'[m, n]$, means something quite different from σA (the latter ‘moves’ whilst the former does not). For example, whilst $\sigma A_0 = A_1 = A'_0$, in general $\sigma A \neq A'$, as seen from $\sigma A_1 = A_2 \neq A'_1$. Indeed a shape A is *stationary* iff $\sigma A = A'$.

(b) We have assumed that A is ‘the set of all occupied cells’. Otherwise, although σA is well defined by (3), it need not have an interpretation in terms of movement in the plane. We could have chosen to impose that condition on A later, but have chosen to facilitate the physical interpretation from the start. \square

To express the relationship between σA_3 and A_0 some ‘domain-specific’ (in this case study, geometric) results are helpful. The need arises simply because two dimensions are a little more complicated than one.

3.2 Helpful geometric results

The function ρ that reflects the plane in the anti-diagonal through the origin is

$$\begin{aligned} \rho : \mathbb{Z}^2 &\rightarrow \mathbb{Z}^2 \\ \rho(m, n) &:= (-n, -m) \end{aligned}$$

and it is lifted pointwise to subsets of the plane.

For any $(k, l) : \mathbb{Z}^2$, the function $\tau_{(k, l)}$ that translates the plane by (k, l) is

$$\begin{aligned} \tau_{(k, l)} : \mathbb{Z}^2 &\rightarrow \mathbb{Z}^2 \\ \tau_{(k, l)}(m, n) &:= (m + k, n + l) \end{aligned}$$

and again it is lifted pointwise to subsets of the plane.

For example we can now see pictorially, from Figure 3, how to identify σA_3 (a proof is given in Theorem 2),

$$(11) \quad \sigma A_3 = \tau_{(1,-1)} A_0$$

and observe two further relationships (self-evident because they do not involve state transition) between the A_i (with \circ for functional composition):

$$(12) \quad A_2 = (\tau_{(1,0)} \circ \rho) A_0$$

$$(13) \quad A_3 = (\tau_{(1,0)} \circ \rho) A_1.$$

Useful straightforward geometric properties are as follows.

Theorem 1. Writing \circ for functional composition,

$$1. \quad \tau_{(k_0,l_0)} \circ \tau_{(k_1,l_1)} = \tau_{(k_0+k_1,l_0+l_1)}$$

$$2. \quad \sigma \circ \tau_{(k,l)} = \tau_{(k,l)} \circ \sigma$$

$$3. \quad \sigma \circ \rho = \rho \circ \sigma$$

$$4. \quad \tau_{(k,l)} \circ \rho = \rho \circ \tau_{(-l,-k)}$$

$$5. \quad (\rho \circ \rho)(m, n) = (m, n).$$

Proof. The first property follows straight from the definition:

$$\begin{aligned} & (\tau_{(k_0,l_0)} \circ \tau_{(k_1,l_1)})(m, n) \\ & = \hspace{20em} \text{definition, twice} \\ & (m+k_1+k_0, n+l_1+l_0) \\ & = \hspace{20em} \text{arithmetic} \\ & (m+k_0+k_1, n+l_0+l_1) \end{aligned}$$

$$= \tau_{(k_0+k_1, l_0+l_1)}(m, n). \quad \text{definition}$$

Since the shape of neighbourhoods, and adjacency, is translation invariant,

$$\tau_{(k,l)} \circ \sigma = \sigma \circ \tau_{(k,l)}.$$

So the second property holds. But similarly

$$\rho \circ \sigma = \sigma \circ \rho$$

and so the third property follows too.

The fourth property follows by naive calculation:

$$\begin{aligned} & (\tau_{(k,l)} \circ \rho)(m, n) \\ &= \tau_{(k,l)}(-n, -m) && \text{definition of } \circ \text{ and } \rho \\ &= \tau_{(-n+k, -m+l)} && \text{definition of } \tau \\ &= \tau_{(-(n-k), -(m-l))} && \text{arithmetic} \\ &= \rho(m-l, n-k) && \text{definition of } \rho \\ &= \rho \circ \tau_{(-l, -k)}(m, n). && \text{definition of } \tau \text{ and } \circ \end{aligned}$$

Finally the last property is trivial. □

3.3 Applying geometry

Let us establish the remaining results of Figure 3: Laws (8) to (11).

Theorem 2. Laws (8) to (11) hold.

Proof. For Law (8) refer to Figure 2, in which cells are labelled using the a, b, c notation from Figure 1. The subsequent state of each cell labelled either a or \bar{b} is unoccupied and so, in particular, the complement

of $N(0,0)$ remains unoccupied. Furthermore, from Figure 1 and the labels of cells interior to that array, Law (8) follows.

Similar reasoning establishes Law (9).

For Law (10),

$$\begin{aligned}
& \sigma A_2 \\
& = && \text{Law (12)} \\
& \sigma((\tau_{(1,0)} \circ \rho)A_0) \\
& = && \text{Theorem 1 (2),(3)} \\
& (\tau_{(1,0)} \circ \rho)(\sigma A_0) \\
& = && \text{Law (8)} \\
& (\tau_{(1,0)} \circ \rho)A_1 \\
& = && \text{Law (13)} \\
& A_3.
\end{aligned}$$

Finally, Law (11) is roughly similar

$$\begin{aligned}
& \sigma A_3 \\
& = && \text{Law (13)} \\
& \sigma((\tau_{(1,0)} \circ \rho)A_1) \\
& = && \text{Theorem 1 (2),(3)} \\
& (\tau_{(1,0)} \circ \rho)(\sigma A_1) \\
& = && \text{Law (9)} \\
& (\tau_{(1,0)} \circ \rho)A_2 \\
& = && \text{Law (12)} \\
& (\tau_{(1,0)} \circ \rho)((\tau_{(1,0)} \circ \rho)A_0) \\
& = && \text{definition of } \circ \\
& (\tau_{(1,0)} \circ \rho \circ \tau_{(1,0)} \circ \rho)A_0 \\
& = && \text{Theorem 1 (4)} \\
& (\tau_{(1,0)} \circ \rho \circ \rho \circ \tau_{(0,-1)})A_0 \\
& = && \text{Theorem 1 (5)} \\
& (\tau_{(1,0)} \circ \tau_{(0,-1)})A_0 \\
& = && \text{Theorem 1 (1)}
\end{aligned}$$

$$\tau_{(1,-1)}A_0.$$

□

3.4 Headings

Following the approach of Section 2, for $(m, n) : \mathbb{Z}^2$ we let

$$x[m, n, t] : \mathbb{B}$$

denote the state of a cell at time $t : \mathbb{N}$.

We use time t to specify desired behaviour, but use cells updated by transition rules (*i.e.*, cellular automata) for implementations. Refinement reasoning leads us from one to the other. The following notation suffices to describe the simple temporal behaviours we are concerned with here.

1. A *rectangle* in the plane is a Cartesian product of two finite intervals $[m_0, n_0)$ and $[m_1, n_1)$:

$$[m_0, n_0) \times [m_1, n_1) = \{(i, j) : \mathbb{Z}^2 \mid m_0 \leq i < m_1 \wedge n_0 \leq j < n_1\}.$$

A subset B of the plane is said to be *bounded* iff it is contained in some rectangle.

If $B \subseteq \mathbb{Z}^2$ is bounded then $rect(B)$ denotes the smallest rectangle containing B . A containing rectangle exists because B is bounded, and the smallest one exists because the set of all rectangles containing any set is closed under intersection.

For example from their definitions (4) to (7) we see

$$\begin{aligned} rect(A_0) &= rect(A_1) = N(0, 0) = [-1, 2) \times [-1, 2) \\ rect(A_2) &= rect(A_3) = N(1, 0) = [0, 3) \times [-1, 2). \end{aligned}$$

Also, from the definition of τ we have

$$rect(\tau_{(1,-1)}A_0) = N(1, -1) = [0, 3) \times [-2, 1)$$

and so infer a property that is useful for our refinement in Section 3.6:

$$(14) \quad A_1 \cup A_2 \cup A_3 \subseteq rect(A_0 \cup \tau_{(1,-1)}A_0) = [-1, 3) \times [-2, 2).$$

2. A bounded subset A (of occupied cells) is said to have *heading* $h(n, k, l)$, where $n : \mathbb{N}$ and $k, l : \mathbb{Z}$, iff for each $t : \mathbb{N}$, after nt time steps A has ‘moved’ by vector (kt, lt) , and moreover at intermediate times

$u \in (nt, n(t+1))$, $\sigma^u A$ lies within the smallest rectangle containing $\sigma^{nt} A$ and $\sigma^{n(t+1)} A$:

$$(15) \quad \sigma^{nt} A = \tau_{(kt, lt)} A$$

$$(16) \quad nt < u < n(t+1) \Rightarrow \sigma^u A \subseteq \text{rect}(\sigma^{nt} A \cup \sigma^{n(t+1)} A).$$

In that case we write $A \in h(n, k, l)$.

This definition is important because (15) relates the n -fold transition (on the left) to a simple translation (on the right). As a result, the union on the right of (16) can also be rewritten as a union of translations. Typically n is the (finite) period of the finite state machine formed by A , in which case it suffices to replace that implication in (16) by its special case $t = 0$.

For example if the (one-dimensional) simple glider were to be embedded in the integer plane via the natural identification of the integers with

$$\{(n, 0) : \mathbb{Z}^2 \mid n \in \mathbb{Z}\}$$

then the simple glider (but with one-dimensional rules) would have heading $h(1, 1, 0)$. The usual glider of the Game of Life (whose first four steps are given in Figure 3 and to which we come next) is stationary every second time step, and when it moves does so alternatively (say) right and down; it thus has heading $h(4, 1, 1)$. Evidently such a glider can be rotated to produce gliders with the other three diagonal headings of ‘magnitude’ 4.

Not all headings are feasible for the Game of Life whose rules ensure that a cell cannot move further than one position each time step. (That does not hold for variants in which, for example, the next state of a cell depends on the current states of cells more distant than its immediate neighbours.) For example $h(1, 2, 2)$ is not feasible. A necessary condition for $h(n, k, l)$ to be feasible is that

$$|k/n|, |l/n| \leq 1.$$

Sufficiency is more subtle; for instance $h(1, 1, 1)$ does not seem possible for a 3×3 set A .

3.5 Two-dimensional gliders

In two dimensions, shapes that move diagonally are called ‘gliders’ (compared, for example, with ‘spaceships’ which move orthogonally [2], page 821). Our specification of a glider is a little more abstract than its implementation, because of the actual—at first sight, slightly erratic—stepwise behaviour of the implementation. However we follow the methodology of Section 2 to reach an implementation by refinement.

A (two-dimensional) *glider* is defined to consist of a subset A of $N(0, 0)$ having some number of occupied

cells and heading $h(4, 1, -1)$:

$$(17) \text{ glider}(A) := \left(\begin{array}{l} A \subseteq N(0,0) \\ 0 < \#\{(m,n) \in A \mid x[m,n]\} \\ A \in h(4, 1, -1) \end{array} \right).$$

The definition allows A , $\sigma^1 A$, $\sigma^2 A$ and $\sigma^3 A$ to have different numbers of occupied cells, provided they lie within the rectangle $\text{rect}(A \cup \sigma^4 A)$.

In the *standard* implementation of a glider, the number of occupied cells remains constant. A stronger specification reflecting that property is obtained by strengthening the second conjunct in (17) with

$$\forall t : \mathbb{N} \cdot \#\{(m,n) \in \sigma^t A \mid x[m,n]\} = \#\{(m,n) \in A \mid x[m,n]\}.$$

In the case of the actual glider, that constant is 5 and the bounding rectangle for the first four steps is given by (14).

3.6 Refinement

We have in mind an implementation that is a \mathbb{Z}^2 cellular automaton in which each cell obeys the transition rules of the Game of Life, (2). As in the one-dimensional case we represent the implementation with an initial condition

$$\text{init} := \forall(m,n) : \mathbb{Z}^2 \cdot x[m,n] = ((m,n) \in A_0),$$

where A_0 is given by (4), and an initialised loop that updates all cells synchronously

$$\text{ca2} := \text{init} \text{ ; } \mathbf{do} \text{ true} \rightarrow \forall(m,n) : \mathbb{Z}^2 \cdot x[m,n] := x'[m,n] \mathbf{od}.$$

Our top-down refinement begins with simple calculus:

$$\text{glider}(A)$$

=

Definition (17)

$$\left(\begin{array}{l} A \subseteq N(0,0) \\ 0 < \#\{(m,n) \in A \mid x[m,n]\} \\ A \in h(4, 1, -1) \end{array} \right)$$

=

Definitions (15, 16)

$$\begin{aligned}
& \left(\begin{array}{l} A \subseteq N(0,0) \\ 0 < \#\{(m,n) \in A \mid x[m,n]\} \\ \forall t: \mathbb{N} \cdot \left(\begin{array}{l} \sigma^{4t}A = \tau_{(t,-t)}A \\ 4t < u < 4(t+1) \Rightarrow \sigma^u A \subseteq \text{rect}(\sigma^{4t}A \cup \sigma^{4(t+1)}A) \end{array} \right) \end{array} \right) \\
& \Leftarrow \text{calculus, with Definition (4)} \\
& \left(\begin{array}{l} A = A_0 \\ \forall t: \mathbb{N} \cdot \left(\begin{array}{l} \sigma^{4t}A = \tau_{(t,-t)}A \\ 4t < u < 4(t+1) \Rightarrow \sigma^u A \subseteq \text{rect}(\sigma^{4t}A \cup \sigma^{4(t+1)}A) \end{array} \right) \end{array} \right) \\
& = \text{calculus (noting that } (\sigma^{4t}A = \tau_{(t,-t)}A_0)[0/t] \text{ is } A = A_0) \\
& \forall t: \mathbb{N} \cdot \left(\begin{array}{l} \sigma^{4t}A = \tau_{(t,-t)}A_0 \\ \forall i: \{1,2,3\} \cdot \sigma^{4t+i}A \subseteq \text{rect}(\sigma^{4t}A_0 \cup \sigma^{4(t+1)}A_0) \end{array} \right) \\
& = \text{calculus} \\
& \forall t: \mathbb{N} \cdot \left(\begin{array}{l} \sigma^{4t}A = \tau_{(t,-t)}A_0 \\ \forall i: \{1,2,3\} \cdot \sigma^{4t+i}A \subseteq \text{rect}(\tau_{(t,-t)}A_0 \cup \tau_{(t+1,-(t+1))}A_0) \end{array} \right) \\
& \Leftarrow \text{calculus, with Definition (14)} \\
& \forall t: \mathbb{N} \cdot \left(\begin{array}{l} \sigma^{4t}A = \tau_{(t,-t)}A_0 \\ \forall i: \{1,2,3\} \cdot \sigma^{4t+i}A = \tau_{(t,-t)}A_i \end{array} \right) \\
& = \text{calculus} \\
& \forall t: \mathbb{N} \cdot \sigma^t A = \tau_{(t \bmod 4, -t \bmod 4)} A_{t \bmod 4}.
\end{aligned}$$

The remainder of the refinement is performed as an induction over time as in the one-dimensional case. When $t = 0$, the invariant above simplifies to

$$A = A_0$$

For the inductive case, we argue with the invariant in the following equivalent form to reflect the four cases of $i = t \bmod 4$:

$$(18) \quad \sigma^{4t+i}A_0 = \tau_{(t,-t)}A_i.$$

Assuming (18) with arbitrary t and i , for the case $t + 1$ we have

$$\begin{aligned}
& \sigma^{4(t+1)+i}A_0 \\
& = \text{calculus} \\
& \sigma^4 \circ \sigma^{4t+i}A_0 \\
& = \text{induction hypothesis (18)}
\end{aligned}$$

$$\begin{aligned}
& \sigma^4 \circ \tau_{(t,-t)} A_i \\
& = \text{Theorem 1 (2)} \\
& \tau_{(t,-t)} \circ \sigma^4 A_i \\
& = \text{Laws (8) to (11)} \\
& \tau_{(t,-t)} \circ \tau_{(1,-1)} A_i \\
& = \text{Theorem 1 (1)} \\
& \tau_{(t+1,-(t+1))} A_i
\end{aligned}$$

as required for (18) with substitution of $t+1$ for t .

4 Conclusion

Polack and Stepney [9] are not the only authors to have claimed that it is not possible to do what we have done. In the multi-agent systems field, Zambonelli and Omicini [15] as well as De Wolf and Holvoet [4] make similar claims based on arguments of Wegner [13]. Wegner, using Gödel's incompleteness theorem, argues that models of interactive systems are necessarily incomplete and therefore that proofs of the nonexistence of incorrect behaviour are generally impossible. The argument, however, applies equally to single-component reactive systems as it does to multi-agent systems. Hence, this is not a new problem, and has not precluded the successful use of formal methods for reactive systems.

Edmonds and Bryson [5] have also argued that formal methods, and in particular refinement, are not relevant to multi-agent systems. Their argument is based on the undecidability of the refinement process. This does not preclude refinement, however, only its automation. Again their argument applies to systems other than multi-agent systems to which formal methods have been applied.

Finally, Bedau [1] defines weak emergence as global behaviour that can be derived from the states and interactions of the components of a system "but only by simulation." Does simulation occur in our development? Only in part of the proof of Laws (8) and (9); the other laws are established by geometric reasoning. It is important to note that even in proving (8) and (9), we have done something more abstract than simulation: for example most cells in the complement of $N(0,0)$ are reasoned about the same way; and those within it are divided into classes to abbreviate the reasoning (Figure 1). Simulation proceeds by brute-force application of the rules; our reasoning does little more than apply their definition.

Our development (refinement) has been leisurely; the essential points fit comfortably on one page. We have been careful to establish every property just in order to explore what exactly is required if a systematic (rather than simulation-based) approach is taken. The fact that we have succeeded is not surprising because of the way that 'emergent functionality' of a loop can be reasoned about using a loop invariant and so does *not* require simulation of the loop. That, in fact, is the basis of the approach taken here. It works because the cellular automaton can be expressed as a loop, and its emergent behaviour reasoned about using the loop invariant.

By making time explicit in the specification, it is possible to describe emergent behaviour which is not expressible at the level of abstraction of the unilateral behaviour of the cells (which does not include time).

How does this extend to other systems with emergent behaviour? Gruner [7] has shown that generalised cellular automata (GCA) where cells may have differing numbers of neighbours, and more states than just on and off, can be used to model mobile agent systems. While our current approach could be extended for GCA, we could also look to the wealth of experience, and formal techniques, for systolic algorithms [10]. A systolic algorithm relies on steadily-moving data being at the right place at the right time. Hence, such techniques could inform an approach to the temporal behaviour of GCA.

Also, Cucker and Smale [3] have recently provided a mathematical proof for the emergence of bird swarming behaviour, and we have presented a refinement of the emergent self-organising behaviour of an algorithm for modular robots [12]. These developments indicate that refinement is indeed more widely applicable to systems with emergent behaviour.

References

- [1] M. A. Bedau. Downward causation and autonomy in weak emergence. *Principia*, **6**:5–50, 2003.
- [2] E. R. Berlekamp, J. H. Conway and R. K. Guy. *Winning Ways for your Mathematical Plays*, volume 2, *Games in Particular*. Academic Press, London, 1982.
- [3] F. Cucker and S. Smale. On the mathematics of emergence. *Japanese Journal of Mathematics*, **2**:197–227, 2007.
- [4] T. De Wolf and T. Holvoet. Towards a methodology for engineering self-organising emergent systems. In H. Czap, R. Unland, C. Branki and H. Tianfield, editors, *Self-Organization and Autonomic Informatics (I)*, volume **135** of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2005.
- [5] B. Edmonds and J. Bryson. The insufficiency of formal design methods—The necessity of an experimental approach for the understanding and control of complex MAS. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, 938–945. IEEE Computer Society, 2004.
- [6] M. Gardner. Mathematical games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, **223**:120–123, 1970.
- [7] S. Gruner. Mobile agent systems and cellular automata. *Autonomous Agent Multi-Agent Systems*, Online First. Springer-Verlag, 2009.
<http://www.springerlink.com/content/c3m2m85p3h678825/fulltext.pdf>
- [8] J. Hu, Z. Liu, G. M. Reed and J. W. Sanders. Ensemble engineering and emergence. In [14], 162–178, 2008.

-
- [9] F. Polack and S. Stepney. Emergent properties do not refine. *Electronic Notes in Theoretical Computer Science*, **137**:163–181, 2005.
- [10] P. Quinton and Y. Robert. *Systolic Algorithms and Architectures*. Prentice-Hall, 1990.
- [11] J. W. Sanders and G. Smith. Formal ensemble engineering. In [14], 132–138, 2008.
- [12] G. Smith and J. W. Sanders. Formal development of self-organising systems. In J. González Nieto, W. Reif, G. Wang and J. Indulska, editors, *International Conference on Autonomic and Trusted Computing (ATC-09)*, volume **5586** of LNCS, 90–104. Springer-Verlag, 2009.
- [13] P. Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, **40**(5):80–91, 1997.
- [14] M. Wirsing, J.-P. Banâtre, M. Hözl and A. Rauschmayer, editors, *Software-Intensive Systems and New Computing Paradigms: Challenges and Visions*, volume **5380** of LNCS. Springer Verlag, 2008.
- [15] F. Zambonelli and A. Omicini. Challenges and research directions in agent-oriented software engineering. *Autonomous Agents and Multi-Agent Systems*, **9**(3):253–283, 2004.