Emergence and refinement

J. W. Sanders[†,1] and Graeme Smith[‡,2]

[†]International Institute for Software Technology, United Nations University, Macao, SAR China;
[‡]School of Information Technology and Electrical Engineering, The University of Queensland, Australia

Abstract. Emergent behaviour—system behaviour not determined by the behaviours of system components when considered in isolation—is commonplace in multi-agent systems, particularly when agents adapt to environmental change. This article considers the manner in which Formal Methods may be used to authenticate the trustworthiness of such systems. Techniques are considered for capturing emergent behaviour in the system specification and then the incremental refinement method is applied to justify design decisions embodied in an implementation. To demonstrate the approach, one and two-dimensional cellular automata are studied. In particular an incremental refinement of the 'glider' in Conway's Game of Life is given from its specification.

Keywords: Emergence, refinement, cellular automata, Game of Life.

1. Introduction

As information systems become even more distributed, interactive and adaptive to uncertain environments, their engineering becomes more subtle. The InterLink programme [WBHR08], for example, has identified an important class of such systems it calls 'ensembles' and has highlighted the need to be able to engineer them in an accountable manner. The difficulty arises because such systems exhibit *emergent behaviour*: system behaviour that is not inferred from the behaviours of system components when considered unilaterally.

'Formal Methods' may be seen as the study of rigorous techniques to account, mathematically and hence with a high degree of trust, for system behaviour. But to date they have not been exercised on the class of systems exhibiting emergent behaviour. Part of the reluctance has no doubt arisen from experience with Complex Systems in which research has focussed on the analysis of *existing* systems (both natural, like the mind, and man-made, like the Internet), rather than the synthesis of new ones, to predict their global properties. Firstly, it may be unclear what precisely constitutes the emergent behaviour (like consciousness, in the case of the mind). Secondly, when modelling an existing system, 'discontinuities' in behaviour may not be known and hence may not be modelled; so proof techniques may not successfully reveal emergent behaviour. Thirdly, it has been claimed that some complex systems exhibit *strong emergence* [Bed03] (*e.g.*, the mind) and therefore, by definition, proofs cannot be constructed of how their behaviour arises.

[1] J.W. Sanders 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.
[2] Graeme Smith acknowledges the support of Australian Research Council (ARC) Discovery Grant DP110101211.

But when we engineer *new* systems, 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 valid implementation. An implementation may in general contain many local variables not present in the specification, and the manner in which they are used may be far from clear. Hence some form of authentication is required. One method is a proof of correctness: a proof that the implementation refines its specification. However a method that uses more of the software engineer's work, and so provides much more information, is an incremental refinement from specification to implementation (since refinement is transitive, correctness is a special case of incremental refinement). Intermediate steps correspond to the engineer's design decisions, like the introduction of local variables, data refinements and algorithms for achieving parts of a computation. Each appears as a refinement step, justified formally by a (mathematical) reason annotating the refinement. Different design decisions lead (in general) to different implementations. For example a sort procedure is specified as permuting its input array so that finally it is (weakly) increasing. But different refinement decisions lead to quite different kinds of sorting algorithms.

Thus it is the chain of reasons for the refinements that authenticate the implementation. By spanning specification to implementation they also authenticate each design decision made in development. So if maintenance, for example, necessitates reworking the development from some point on, only from that point need the designs and their justifications be changed. An incremental refinement can be seen as a way of structuring a correctness proof to reflect engineering practice. Even better, it provides a way to understand an implementation at any level of abstraction in the hierarchy of refinements, by following the refinements to just that level.

In spite of all its benefits, incremental refinement does not explain where the design steps 'come from'. It states each step formally and justifies the refinement mathematically, but does not motivate it. In that way it is (like) a mathematical proof. When teaching program development using refinement, steps are typically motivated by (informal) appeals to computational complexity (time, space) and to the goal-directed nature of the implementation being code. For example, for evaluation of a polynomial $\sum_{0\leq j<n} a_j x^j$ at a given point x_0, the specification would assign to the final variable the value $\sum_{0\leq j<n} a_j x_0^j$. A refinement step might bracket that sum so as to use only a linear, rather than a quadratic, number of multiplications (Horner's rule). Justification for the step is trivial: distributivity of multiplication over addition. But motivation comes from the desire for a program whose asymptotical complexity is linear rather than quadratic.

A further part of the reluctance to use Formal Methods for systems exhibiting emergence may well be the seeming inconsistency between emergence and reductionism: how can an incremental refinement from a specification containing emergent behaviour result in an implementation whose components have unilateral behaviours that are insufficient to account for the emergent behaviour? The resolution is simple: the emergent behaviour results from inter-component interactions. But, as already stated, the synthesis of such systems is far from simple: the emergent behaviour must be shown to be a consequence of those interactions. Computer Science, and this paper in particular, is interested in systems where that is the case, *i.e.*, systems which exhibit *weak emergence* [Bed03]. 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: is Formal Methods, and in particular the technique of incremental refinement, applicable to the engineering of systems with (weak) emergent properties? There are claims that it is not; they are discussed in Section 6. One paper, [PS05], posits that the emergent behaviour of the 'glider' pattern of Conway's Game of Life[3] [Gar70, BCG82] cannot be authenticated by incremental refinement. The present paper shows that it can.

Conway's Game of Life is a 2-dimensional cellular automaton which simulates the evolution of an infinite grid of cells. The cells evolve according to the following four rules, where the neighbours of a cell are the eight cells surrounding it.

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.

[3] An executable version can be found at `http://www.math.com/students/wonders/life/life.html`.

Many different patterns can be formed in the Game of Life including dynamic patterns such as the glider which translates itself across the grid.

In this paper 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. A less-complete version has appeared in [SS09a]. The refinement calculus [Mor94] that we use to affect the incremental refinements is summarised in Section 2. The refinement calculus provides a formalism (*i.e.*, notation and laws) for performing incremental refinements. But before it can be applied to the glider in the Game of Life the glider's emergent behaviour must be specified. In Section 3 we consider techniques to achieve that, both in general and in the Game of Life in particular. Then, to demonstrate our approach on an example with fewer distracting details than the 2-dimensional Game of Life, in Section 4 we first consider a 1-dimensional cellular automaton and an emergent behaviour it exhibits. To show that in making that simplification we have not inadvertently brought together the two levels of abstraction, in Section 5 we provide a specification and refinement of the glider in the full 2-dimensional Game of Life. The approach is the same; only the complexity of the detail is increased. In Section 6 we discuss work related to the question of refining emergent properties and in Section 7 conclude.

2. Stepwise refinement

The purpose of this section is to summarise standard material used later in the paper to perform incremental refinements, and to emphasise how the notation handles the distinction between functional and non-functional properties.

The specification statement (see Morgan's textbook [Mor94][4])

$$x : [pre(x), post(x, x')]$$

denotes a computation that by changing only variables in the list x terminates, whenever begun in a state satisfying the predicate *pre*, in a state satisfying the binary predicate *post*. The postcondition, with free variables x for initial state and x' for final state, need not be executable. Indeed the purpose of a specification is to describe succinctly the result of the computation and not the manner in which it is executed; that is the purpose of the implementation. For example if variable n is an integer then

$$n : [n \geq 0, n' > n] \tag{1}$$

is satisfied by just those computations that terminate when begun in a state for which $n \geq 0$, and do so in a state having a greater value of n than initially. It is guaranteed to terminate only if $n \geq 0$; and the postcondition is nondeterministic, satisfied by any larger final value of n.

A computation refines another iff it is at least as deterministic. Recall that weakening of the precondition *pre* and strengthening of the postcondition *post* ensures refinement between specification statements:

$$x : [pre, post] \sqsubseteq x : [pre1, post1] \quad \text{if} \quad pre \Rightarrow pre1 \text{ and } pre \wedge post1 \Rightarrow post\,.$$

For example the statement (1) is refined by both $n : [true, n' = n + 1]$ and $n : [n \geq -1, n' = n + 2]$, but by neither $n : [true, n' = n - 1]$ nor $n : [n > 0, n' = n + 1]$.

Recall that the semantics of assignment yields the law by which assignments are introduced into an incremental refinement. In its simplest form, for an expression e that is always well defined

$$x : [true, x' = e] \;=\; x := e\,.$$

An implementation describes the manner in which a computation is to be executed. In this paper particularly simple cellular automata are considered, which it suffices to describe using nonterminating loops of assignments to the variables local to each cell; initialisation is achieved again by assignment to all local variables. To describe such assignments we use the guarded-command language, although the loops containing them are nonterminating, reflecting the unending action of the automaton.

Of particular importance in this paper is the special case of an initialised loop that governs the unending behaviour of a cellular automaton. If the loop invariant is given at one level of abstraction, a refinement is

[4] We differ slightly by writing x rather than x_0 for initial values and x' rather than x for final values.

required at the next level in which the loop body is refined closer to code. If the invariant is deterministic then the loop body is determined by having to maintain it.

A simple example is provided by repeated addition to sum the numbers in an array. For a natural number N, an array $a : array[0, N)$ of integers, and an integer s, this specification results in s equalling the sum of the values in a

$$s : [true, s' = \sum_{0 \le j < n} a[j]] \,.$$

The postcondition is not code if that sum is not atomically evaluated. But code may be achieved by repeatedly adding elements of the array. Such a refinement may be achieved by introducing a local variable i of natural number type to be the array index, and requiring a loop with guard $i < N$ to maintain the invariant $inv \;:=\; (0 \le i \le N) \wedge (s = \sum_{0 \le j < i} a[j])$. It then follows that with i initially 0, for the invariant to be established s must also initially be 0 (because the empty sum is 0). The refinement step is justified because the negation of the guard and the invariant together imply the specification's postcondition. Writing inv' for the predicate $inv[i', s'/i, s]$,

var $i : \mathbb{N}$ ·
  $i, s := 0, 0$ ⨾
  do $i < N \;\rightarrow$
    $i, s : [inv \wedge i < N, inv']$
  od
rav

We suppose that progress is to be achieved in the loop by incrementing i. Then the body of the loop is refined by strengthening its postcondition

$$i, s : [inv \wedge i < N, inv' \wedge i' = i + 1]]$$

(where termination is guaranteed by the strict decrease, on each iteration, of the variant function $N - i$). Predicate calculation then shows that refinement holds provided

$$i, s := i + 1, s + a[i] \,,$$

which completes an outline of an incremental refinement to code. The steps are collected to provide an incremental refinement in the Appendix. Further examples appears in Sections 4 and 5.

If, alternatively, initialisation were $i := N - 1$ and the invariant were $(0 \le i \le N) \wedge (s = \sum_{i \le j < N-1} a[j])$ then progress would be achieved by decrementing i and a different implementation would result. Naturally the collection of terms in other ways leads to other designs.

In our case, the refinements start from a specification that includes a description of emergent behaviour, and from there progress to an implementation by using laws of the refinement calculus. The approach succeeds because the emergent behaviour is captured functionally (as described in Section 3). Many treatments of emergence in the History and Philosophy of Science require that the emergent behaviour be attended by some extra-functional 'element of surprise' (see, for example, the editorial [Dam00] by Damper). Here we follow the standard Scientific approach in ignoring that aspect, which depends on the state of mind of the observer. That is why incremental refinement, using just functional properties, suffices.

Can the two approaches (Scientific and Philosophical) be reconciled? A treatment incorporating that less quantifiable 'element of surprise' would have to be couched in terms of motivations for the various refinement steps, requiring the inclusion of a motivation for each step. In the example of polynomial evaluation (from Section 1), for instance, the motivation is the complexity advantage provided by Horner's rule. The augmented incremental refinement would make it clear what each step contributes by way of 'surprise'. Emergence predicates in adaptive multi-agent systems (AMAS) often require intricate designs in their incremental refinement, as do efficient distributed systems in general. If a system specifier is over-optimistic then the emergence-enriched specification will be infeasible: not refineable to code. Thus the identification of an emergence predicate is by itself not enough to guarantee that an implementation exists. Incremental refinement establishes it, and if steps are augmented with extra-functional motivations (like complexity arguments) then it also highlights designs of 'surprise'. We do not pursue this further.

However even in purely functional form, the incremental refinement of systems exhibiting emergence is

of interest because of claims that it is impossible. To achieve an incremental refinement, the first step must be the capturing of the emergent behaviour in the specification. We turn to that now.

3. Specifying emergence

The purpose of this section is to consider what is required in enriching a specification to capture emergent behaviour.

A typical situation seems to be a multi-agent system (MAS) whose emergent behaviour results from inter-agent interactions that are not predictable from consideration of the agents in isolation. Suppose that each agent i $(0 \leq i < n)$ has unilateral specification A_i. Then a 'naive' specification for the system is the conjunction

$$\bigwedge_{0 \leq i < n} A_i \,.$$

It is naive by being restricted to unilateral behaviours only: each conjunct is expressed entirely in terms of its own local state. However it is made realistic by enriching it with emergent behaviour, done by using global variables to express what interagent interactions achieve. That extra conjunct, *emerge*, has been called an *emergence predicate* [HLRS08] and results in the emergence-enriched specification

$$\mathit{emerge} \wedge \bigwedge_{0 \leq i < n} A_i \,.$$

The difficulty lies, of course, in formulating *emerge*. But once that has been achieved, incremental development from the emergence-enriched specification is possible. Let us consider some examples of emergence predicates, with just enough detail to appreciate the kind of global information they require.

One of the most popular examples of emergent behaviour is the flocking of birds. For most of the day each bird i functions autonomously, so A_i is expressed in terms of i's local state and environmental interactions (save with other birds). But particular inter-bird interactions are required to form and maintain a flock; so *emerge* refers to the states of all birds. It also contains global variables, like the position and velocity of the flock, that do not appear in the (distributed) 'implementation' which consists only of the birds interacting in parallel (none with global information). Details, with correctness of a design in which each bird interacts with its nearest neighbours in the flock by adjusting its own position and velocity to the average of those of its neighbours, are given in [HLRS08] (where a theorem of Cucker and Smale [CS07b] is exploited to ensure that the design establishes *emerge*). This example demonstrates that, since the flock parameters are defined in three dimensional space, *emerge* may well exhibit both discrete and analogue parts. Furthermore, that differentiable methods may be required in establishing refinements.

Statistical methods may also be required in expressing *emerge*. For example each agent i in a MAS may have a Boolean state component b_i. So A_i includes a conjunct constraining b_i in terms of the rest of i's state. Suppose that if an agent i is chosen at random then it is equally likely for b_i to be true or false. That behaviour is emergent because it requires particular coordination between the agents (in a distributed design). Thus *emerge* refers to each b_i and some global tolerance which measures acceptable deviation from $\frac{1}{2}$ of the proportion of agents for which b_i is true. In general, statistical *clichés* may be required in *emerge*, and results like convergence theorems from Statistics required in establishing refinements. For details we refer to [HLRS08].

In the present paper, however, implementations are cellular automata. Each i is a cell and A_i describes the way in which i's state changes as a result of the states of i's immediate neighbours. The automaton thus has a clock that ensures each cell changes state at the same time; but it has no notion of global time. In specifying emergent behaviour, like the movement of shapes, global time is used to constrain local states. Global time thus plays the part of the global variables used in the previous examples.

There is an interesting analogy between an emergence predicate *emerge* and the invariant of a loop in a sequential program. Both use, in general, variables not present in the implementation to express its behaviour. The incremental refinement of a loop from its invariant, as demonstrated in the example of array summation in Section 2, is standard. The major contribution of this paper may be viewed as achieving the

same thing for systems with emergent behaviour. In fact, in the cellular-automaton implementation of the Game of Life, initialisation is followed by a loop whose invariant is simply *emerge*. So the methods coincide.[5]

The claim that an incremental refinement of the glider is not possible thus reduces to the impossibility of either (a) specifying the emergent behaviour of the glider (achieved here by use of global time), or (b) an incremental refinement of a loop from its invariant (achieved here in the refinement calculus). We have indicated why in principle both are possible; we now consider details.

4. One-dimensional cellular automaton

We consider a 1-dimensional cellular automaton on an integer grid of cells. Its evolution rule *could* involve more than immediate neighbours, or depend on more than the previous generation. For several nice examples treated rigorously see [BCF10]. However by analogy with Conway's 2-dimensional rule we limit the rule to just immediate neighbours and just the previous generation and choose it, as follows, to capture behaviour analogous to that in two dimensions.

1. A cell is live in the next generation iff in the present generation exactly one of its two immediate neighbours is live.

The state of the cell itself in the present generation is irrelevant. There are many alternative rules (for example that rule could also require that the cell be dead in the present generation) but that one is typical and satisfies our previous criteria. It is in fact the well known Rule 90 of elementary cellular automata (see `http://mathworld.wolfram.com/Rule90.html`).

As in the planar case, all cells are updated synchronously. We consider state to be a function which assigns a Boolean to each integer: *true* for live (or 'occupied') and *false* for dead (or 'vacant'). Let $\mathbb{B}$ denote the type of Booleans. We write state at the current generation as $x : \mathbb{Z} \to \mathbb{B}$, and its value at the next generation as $x' : \mathbb{Z} \to \mathbb{B}$.

To formalise the update rule we let νx denote, pointwise, the number of live neighbours[6] of x:

$$\begin{aligned} \nu &: (\mathbb{Z} \to \mathbb{B}) \to (\mathbb{Z} \to \mathbb{N}) \\ (\nu x)[n] &:= \#\{m : \mathbb{Z} \mid x[m] \wedge |m - n| = 1\} . \end{aligned} \tag{2}$$

Evidently for any state x, νx takes values between 0 and 2.

Then the rule is

$$\forall\, n : \mathbb{Z} \cdot x'[n] = (\nu x[n] = 1) . \tag{3}$$

Two examples are given in Figure 1.

To emphasise that in each generation the assignments are made simultaneously (for each n) and synchronously, we adopt a functional notation in which the above rule is expressed as

$$x' = (\nu x = 1) . \tag{4}$$

That is, a single assignment is made to the function x; it is interpreted as consisting of the simultaneous and synchronous assignment to each of its values $x[n]$.

In functional form, the state that is *false* at each $n : \mathbb{N}$ except m, is a translated Dirac delta function:

$$\begin{aligned} \delta_m &: \mathbb{Z} \to \mathbb{B} \\ \delta_m &:= \lambda\, n : \mathbb{Z} \cdot (n = m) . \end{aligned}$$

The *support* of state $x : \mathbb{Z} \to \mathbb{B}$ is defined to consist of those integers at which x takes the value *true*:

$$sp\, x := \{n : \mathbb{Z} \mid x[n]\} .$$

The following simple result is used in our development. For a nonempty finite set E of integers we write $\sqcap E$ for the least element of E and $\sqcup E$ for the greatest element.

[5] It is interesting to observe that, conversely, a loop body may be regarded as an agent that updates its state (determined by the loop variables). Thus the loop itself can be viewed as a MAS with a varying number of agents (one for each iteration), depending on the state in which the loop is begun. Then the emergence predicate of the MAS *is* the loop invariant.

[6] The neighbours of a cell $x[n]$ are strictly those cells adjacent to $x[n]$, *i.e.*, $x[n-1]$ and $x[n+1]$, and do not include $x[n]$. This definition differs from that often used in the cellular automata literature which includes cell $x[n]$ in the set of neighbours.

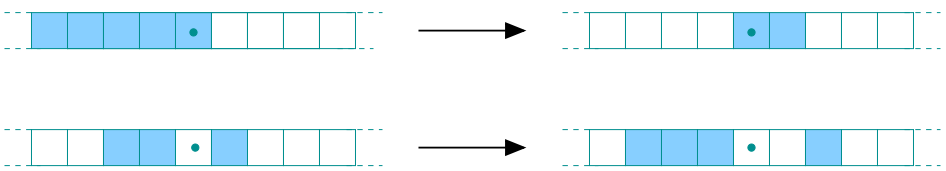

Fig. 1. Two examples of the 1-dimensional rule (4). In the first $x = (n \leq 0)$ and $x' = (n \in \{0, 1\})$. In the second $x = (n \in \{-2, -1, 1\})$ and $x' = (n \in \{-3, -2, -1, 2\})$. The cell at the origin is depicted with a dot.

Theorem 1. Suppose that state x has nonempty finite support. Then, under Rule (4), the least element of the support of x' is one less than the least element of the support of x. Analogously the greatest element is one more.

(i) $\sqcap(sp\, x') \;=\; \sqcap(sp\, x) - 1$
(ii) $\sqcup(sp\, x') \;=\; \sqcup(sp\, x) + 1$

Proof. By de Morgan's law for negation and extrema, namely $\sqcap E = -(\sqcup(-E))$, it suffices to prove the first claim. By calculation:

$$
\begin{array}{ll}
& m = \sqcap(sp\, x) - 1 \\
\equiv & \qquad \text{arithmetic} \\
& m + 1 = \sqcap(sp\, x) \\
\equiv & \qquad \text{definition of minimum} \\
& x[m+1] \wedge \forall\, n \leq m \cdot \neg x[n] \\
\equiv & \qquad \text{calculus} \\
& x[m+1] \wedge \neg x[m-1] \wedge \forall\, n \leq m \cdot \neg x[n] \\
\equiv & \qquad \text{calculus and Definition (2)} \\
& (\nu x)[m] = 1 \;\wedge\; \forall\, n \leq m \cdot \neg x[n] \\
\equiv & \qquad \text{Rule (4) and } k := n - 1 \\
& x'[m] \wedge \forall\, k < m \cdot \neg x[k-1] \wedge \neg x[k+1] \\
\equiv & \qquad \text{Rule (4)} \\
& x'[m] \wedge \forall\, k < m \cdot \neg x'[k] \\
\equiv & \qquad \text{definition of minimum} \\
& m = \sqcap(sp\, x')\,.
\end{array}
$$

The result does not hold if the support is unbounded (with unbounded extrema defined in terms of $\pm\infty$ as usual); a counterexample is provided by the first example in Figure 1. Fortunately our application is to states x having nonempty finite support.

4.1. Specification

To describe 1-dimensional patterns we augment the cellular automaton with a notion of discrete global time that counts the number of generations since initialisation. It is to be emphasised that global time is a specification 'artifact' and not available to the implementation, which is able merely to update, instantaneously, from the current generation to the next. Indeed evaluation of the state of any cell at time $t : \mathbb{N}$ (after initialisation at time $t = 0$) would involve evaluation of the state of cells at distance t away. Since one clock cycle is needed to evaluate the state of cells each unit away, as t tends to infinity so would the time taken. As a result, behaviour of the Game of Life which requires evaluation of the state of a cell at an arbitrary time is not derivable strictly at the implementation level. It is thus, by the definition [HLRS08], emergent

behaviour. Global time is introduced because it enables us to specify the emergent phenomena of interest, but must be removed during the derivation of an implementation.

At any time $t : \mathbb{N}$ we write the Boolean state at $n : \mathbb{Z}$ as

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

A *flood* consists of two cells moving away from the origin, one in each direction, so that at each time step t the cells are at locations $n = \pm t$. Between the cells arbitrary behaviour is allowed but beyond them all cells are dead. It is specified simply using global time $t : \mathbb{N}$ as a free variable:

$$\mathit{flood} \;:=\; x[-t, t] \;\wedge\; x[t, t] \;\wedge\; \forall\, m : \mathbb{Z} \cdot \; |\, m \,| > t \;\Rightarrow\; \neg x[m, t] \,. \tag{5}$$

Notice that the initial state is given by $t = 0$, in which just the cell at the origin is live. For $t = 1$ the cells at $n = \pm 1$ are live but the cell at the origin is undetermined. Specification (5) can be expressed simply in terms of support: at any time t the support of x (as a function of n, with t fixed) has minimum $-t$ and maximum t.

$$\mathit{flood} \;\equiv\; \sqcap \mathit{sp}\,(\lambda\, n : \mathbb{Z} \cdot \; x[n, t]) = -t \quad \wedge \quad \sqcup \mathit{sp}\,(\lambda\, n : \mathbb{Z} \cdot \; x[n, t]) = t \,. \tag{6}$$

This is the form exploited by Section 4.3 in the guise of Theorem 1.

The stronger specifications in which cells between $\pm t$ are all dead, or all live, are infeasible with Rule (4) as shown by simple calculation. A rule which does make it feasible for all interstitial cells to be dead is studied in [SS09a]; however it might be argued that it makes the specification and implementation unnecessarily close, which is why we adopt an alternative here.

4.2. Implementation

The implementation we seek has an initialisation command, *init*, assumed to be an assignment to the function x, which corresponds to the specification with $t = 0$. Subsequently it updates all cells synchronously according to Rule (4), taking one t-time unit to do so. In the guarded-command language it is expressed

$$\begin{array}{lll} \mathit{ca} & := & \mathit{init} \;\fatsemi \\ & & \mathbf{do}\ \mathit{true} \;\rightarrow\; x := x'\ \mathbf{od} \end{array} \tag{7}$$

where the update is given by (4) and the time constraint on its execution is given by

$$\forall\, t : \mathbb{N}, n : \mathbb{Z} \cdot \; x[n, t] = x[n] \;\Rightarrow\; x[n, t{+}1] = x'[n] \,. \tag{8}$$

That representation of a reactive (nonterminating) program in the guarded-command language abstracts the command that displays the state x. That command outputs on each iteration and so saves the program from simply diverging.

4.3. Refinement

Use of the guarded-command language allows us to use the refinement calculus to perform our simple refinements concisely.

Refinement of *init* is achieved by substitution of 0 for t in *flood*:

$$\begin{array}{ll} \mathit{init} & \\ = & \text{specification} \\ x : [\mathit{true}, \mathit{flood}[0/t]] & \\ = & \text{definition of } \mathit{flood}\ (5) \\ x : [\mathit{true}, x[0, 0] \wedge \forall\, m : \mathbb{Z} \cdot \; |\, m \,| > 0 \;\Rightarrow\; \neg x[m, 0]] & \\ = & \text{initialisation assumption, } t = 0 \\ x : [\mathit{true}, \forall\, n : \mathbb{Z} \cdot \; x[n] = (n = 0)] & \\ = & \text{functional notation} \end{array}$$

$$
\begin{array}{ll}
& x : [\mathit{true}, x = \delta_0] \\
= & \qquad \text{semantics of assignment} \\
& x := \delta_0 \,.
\end{array}
$$

It consists, as specified, of the assignment of *false* to each cell except to that at the origin.

We now use the fact that *flood* is the invariant of the loop in the implementation *ca* to infer the loop body.

$$
\begin{array}{ll}
& x : [\mathit{flood}, \mathit{flood}[t{+}1/t]] \\
= & \qquad \text{definition of } \mathit{flood}\ (5) \\
& x : [\ x[-t,t] \wedge x[t,t] \wedge \forall\, m : \mathbb{Z} \cdot\ |\, m \,| > t \Rightarrow \neg x[m,t], \\
& \qquad x[-(t{+}1), t{+}1] \wedge x[t{+}1, t{+}1] \wedge \forall\, m : \mathbb{Z} \cdot\ |\, m \,| > t{+}1 \Rightarrow \neg x[m, t{+}1]\ ] \\
= & \qquad (8) \\
& x : [\ x[-t] \wedge x[t] \wedge \forall\, m : \mathbb{Z} \cdot\ |\, m \,| > t \Rightarrow \neg x[m], \\
& \qquad x'[-(t{+}1)] \wedge x'[t{+}1] \wedge \forall\, m : \mathbb{Z} \cdot\ |\, m \,| > t{+}1 \Rightarrow \neg x'[m]\ ] \\
\sqsubseteq & \qquad \text{laws of the refinement calculus: precondition in postcondition, weaken precondition} \\
& x : [\ \mathit{true}, x[-t] \wedge x[t] \wedge x'[-(t{+}1)] \wedge x'[t{+}1] \wedge \\
& \qquad\qquad \forall\, m : \mathbb{Z} \cdot\ |\, m \,| > t \Rightarrow \neg x[m] \wedge \\
& \qquad\qquad \forall\, m : \mathbb{Z} \cdot\ |\, m \,| > t{+}1 \Rightarrow \neg x'[m]\ ] \\
= & \qquad \text{definition of } \mathit{sp} \\
& x : [\ \mathit{true}, \sqcap \mathit{sp}\, (\lambda\, n : \mathbb{Z} \cdot\ x[n,t]) = -t\ \wedge\ \sqcup \mathit{sp}\, (\lambda\, n : \mathbb{Z} \cdot\ x[n,t]) = t\ \wedge \\
& \qquad\qquad \sqcap \mathit{sp}\, (\lambda\, n : \mathbb{Z} \cdot\ x'[n,t]) = -(t+1)\ \wedge\ \sqcup \mathit{sp}\, (\lambda\, n : \mathbb{Z} \cdot\ x'[n,t]) = t+1\ ] \\
\sqsubseteq & \qquad \text{Theorem 1} \\
& x : [\ \mathit{true}, \nu x = 1\ ] \\
= & \qquad \text{semantics of assignment} \\
& x\ :=\ (\nu x = 1)\,.
\end{array}
$$

Thus in order for *flood* to be invariant, iteration of Rule (4) suffices and the refinement is complete.

Although Rule (4) and the implementation *ca* are simple, they embody the principle being confirmed here: the implementation may be refined stepwise from the emergence-enriched specification.

5. 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 the Game of Life consists again of a function, this time $x : \mathbb{Z}^2 \to \mathbb{Z}^2$, whose state pointwise (or cell-wise) is written $x[m,n]$ and whose state pointwise after a transition is written $x'[m,n]$.

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

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

Evidently $0 \leq \nu(m,n) \leq 8$.[7]

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

[7] In Section 4 (Definition 2) the state argument x of ν was made explicit because we were concerned with conceptual clarity; here we are more concerned with calculational facility and so it is omitted.

label	$x[m,n]$	$\nu(m,n)$	$x'[m,n]$
a		$\neq 2,3$	*false*
b	*true*	2	*true*
c	*true*	3	*true*
$\overline{b}$	*false*	2	*false*
$\overline{c}$	*false*	3	*true*

Fig. 2. Labels for five important types of cell in the Game of Life.

a	a	a	a	a	a	a
a	a	a	$\overline{b}$	a	a	$\boldsymbol{a}$
a	a	a	a	b	$\overline{b}$	a
a	a	$\overline{c}$	a	c	$\overline{b}$	a
a	a	$\overline{b}$	b	$\overline{c}$	a	a
a	a	a	a	a	a	a
a	a	a	a	a	a	a

Fig. 3. An initial configuration of the 2-dimensional glider with cells labelled using the convention of Figure 2.

rules simplify to

$$x'[m,n] \;:=\; \begin{array}{l} \nu(m,n) = 3 \\ \vee \\ \nu(m,n) = 2 \;\wedge\; x[m,n]\,. \end{array} \tag{9}$$

In functional notation, with operations interpreted pointwise, that becomes

$$x' \;:=\; (\nu = 3) \vee (\nu = 2 \;\wedge\; x)\,. \tag{10}$$

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

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

A cell $x[m,n]$ satisfying $\nu(m,n) \neq 2,3$ 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 2 and an example appears in Figure 3.

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:

$$\sigma A := \{(m,n) \mid x'[m,n]\}\,. \tag{11}$$

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 4,

$$A_0 := \{(-1,1),(0,0),(0,-1),(1,0),(1,1)\} \tag{12}$$
$$A_1 := \{(-1,0),(0,-1),(1,-1),(1,0),(1,1)\} \tag{13}$$
$$A_2 := \{(0,1),(0,-1),(1,0),(1,-1),(2,0)\} \tag{14}$$
$$A_3 := \{(0,-1),(1,1),(1,-1),(2,0),(2,-1)\} \tag{15}$$

we have

$$\sigma A_0 = A_1 \tag{16}$$
$$\sigma A_1 = A_2 \tag{17}$$
$$\sigma A_2 = A_3\,. \tag{18}$$

Note that we have assumed A to be 'the set of all occupied cells'. Otherwise, although σA is well defined

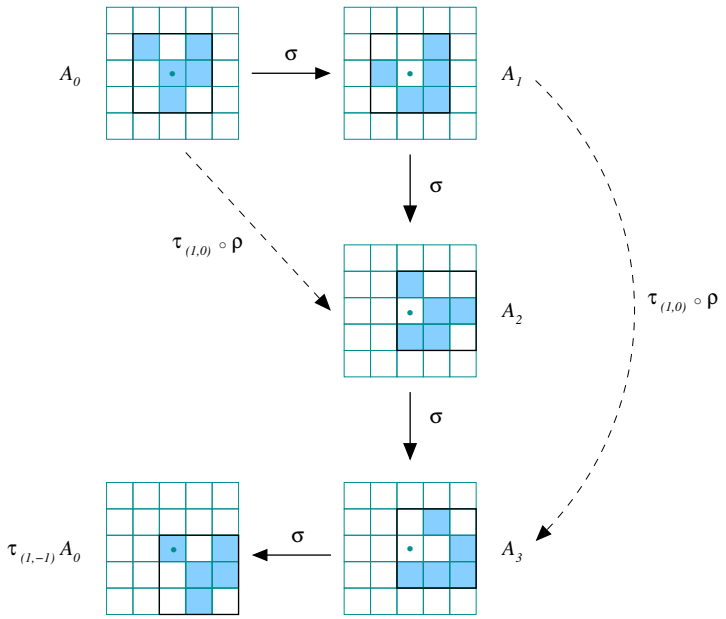

Fig. 4. Relationships between glider configurations. The origin, marked as a dot, lies at the centre of each 5×5 array.

by (11), 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.

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 of the geometrical complexity of two dimensions compared with one.

5.1. Helpful geometric results

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

$$\rho : \mathbb{Z}^2 \to \mathbb{Z}^2$$
$$\rho(m, n) \;:=\; (-n, -m)$$

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

$$\tau_{(k,l)} : \mathbb{Z}^2 \to \mathbb{Z}^2$$
$$\tau_{(k,l)}(m, n) \;:=\; (m + k, n + l)$$

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

For example we can now see pictorially, from Figure 4, how to identify σA_3 in terms of A_0 (a proof is given in Theorem 2),

$$\sigma A_3 = \tau_{(1,-1)} A_0 \tag{19}$$

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

$$A_2 = (\tau_{(1,0)} \circ \rho) A_0 \tag{20}$$
$$A_3 = (\tau_{(1,0)} \circ \rho) A_1 \,. \tag{21}$$

Useful straightforward geometric properties are as follows.

Theorem 2. Writing $\circ$ for functional composition,

1. $\tau_{(k0,l0)} \circ \tau_{(k1,l1)} \;=\; \tau_{(k0+k1,l0+l1)}$
2. $\sigma \circ \tau_{(k,l)} \;=\; \tau_{(k,l)} \circ \sigma$
3. $\sigma \circ \rho \;=\; \rho \circ \sigma$
4. $\tau_{(k,l)} \circ \rho \;=\; \rho \circ \tau_{(-l,-k)}$
5. $(\rho \circ \rho)(m,n) = (m,n)\,.$

Proof. The first property follows straight from the definition:

$$
\begin{array}{ll}
(\tau_{(k0,l0)} \circ \tau_{(k1,l1)})(m,n) & \\
= & \text{definition, twice} \\
(m{+}k1{+}k0,\, n{+}l1{+}l0) & \\
= & \text{arithmetic} \\
(m{+}k0{+}k1,\, n{+}l0{+}l1) & \\
= & \text{definition} \\
\tau_{(k0+k1,l0+l1)}(m,n)\,. &
\end{array}
$$

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. The third property is similar.

The fourth property follows by naive calculation:

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

Finally the last property is trivial. □

5.2. Applying geometry

Let us establish the remaining results of Figure 4: Laws (16) to (19).

Theorem 3. Laws (16) to (19) hold.

Proof. For Law (16) refer to Figure 3, in which cells are labelled using the a, b, c notation from Figure 2. The subsequent state of each cell labelled either a or $\overline{b}$ is unoccupied and so, in particular, the complement

of $N(0,0)$ remains unoccupied. Furthermore, from Figure 2 and the labels of cells interior to that array, Law (16) follows.

Similar reasoning establishes Law (17).

For Law (18),

$$
\begin{array}{ll}
& \sigma A_2 \\
= & \text{Law (20)} \\
& \sigma\left((\tau_{(1,0)} \circ \rho) A_0\right) \\
= & \text{Theorem 2 (2),(3)} \\
& (\tau_{(1,0)} \circ \rho)(\sigma A_0) \\
= & \text{Law (16)} \\
& (\tau_{(1,0)} \circ \rho) A_1 \\
= & \text{Law (21)} \\
& A_3 \,.
\end{array}
$$

Finally, Law (19) is roughly similar

$$
\begin{array}{ll}
& \sigma A_3 \\
= & \text{Law (21)} \\
& \sigma((\tau_{(1,0)} \circ \rho) A_1) \\
= & \text{Theorem 2 (2),(3)} \\
& (\tau_{(1,0)} \circ \rho)(\sigma A_1) \\
= & \text{Law (17)} \\
& (\tau_{(1,0)} \circ \rho) A_2 \\
= & \text{Law (20)} \\
& (\tau_{(1,0)} \circ \rho)((\tau_{(1,0)} \circ \rho) A_0) \\
= & \text{definition of } \circ \\
& (\tau_{(1,0)} \circ \rho \circ \tau_{(1,0)} \circ \rho)) A_0 \\
= & \text{Theorem 2 (4)} \\
& (\tau_{(1,0)} \circ \rho \circ \rho \circ \tau_{(0,-1)})) A_0 \\
= & \text{Theorem 2 (5)} \\
& (\tau_{(1,0)} \circ \tau_{(0,-1)})) A_0 \\
= & \text{Theorem 2 (1)} \\
& \tau_{(1,-1)} A_0 \,.
\end{array}
$$

□

5.3. Specification

Following the approach of Section 4, 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 the former to the latter. 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 $[m0, n0)$ and $[m1, n1)$:

$$[m0, n0) \times [m1, n1) \;=\; \{(i, j) : \mathbb{Z}^2 \mid m0 \leq i < m1 \;\wedge\; n0 \leq j < n1\}\,.$$

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 (12) to (15) we see

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

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 5.4:

$$A_1 \cup A_2 \cup A_3 \subseteq rect(A_0 \cup \tau_{(1,-1)} A_0) \;=\; [-1,3) \times [-2,2)\,. \tag{22}$$

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 $\sigma^{n(t+1)} A$:

$$\sigma^{nt} A \;=\; \tau_{(kt,lt)} A \tag{23}$$

$$nt < u < n(t+1) \Rightarrow \sigma^u A \subseteq rect(\sigma^{nt} A \;\cup\; \sigma^{n(t+1)} A)\,. \tag{24}$$

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

This definition is important because (23) relates the n-fold transition (on the left) to a simple translation (on the right). As a result, the union on the right of (24) 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 (24) by its special case $t = 0$.

The usual glider of the Game of Life (whose first four steps are given in Figure 4 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.

5.3.1. Glider specified

In two dimensions, shapes that move diagonally are called 'gliders'. 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 4 to reach an implementation by refinement.

A *glider* is defined to consist of a subset A of $N(0, 0)$ with a positive number of occupied cells and heading $h(n, k, l)$ for some $n \in \mathbb{N}$ and $k, l \in \mathbb{Z}$:

$$glider(A) \;:=\; \left(\begin{array}{l} \{\,\} \subset A \subseteq N(0,0) \\ \exists\, n : \mathbb{N};\; k, l : \mathbb{Z} \cdot\; A \in h(n, k, l) \end{array} \right)\,. \tag{25}$$

The parameter A is the glider's initial state, made explicit for convenience. Global time t, although it is implicit, is essential to that definition by virtue of (23) and (24). Its purpose, just as in the 1-dimensional case, is to express the emergent behaviour required of the cellular automaton.

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 $rect(A \cup \sigma^4 A)$.

5.4. Refinement

As for one dimension we seek an implementation that is a cellular automaton in which each cell obeys the transition rules; but now the automaton is two dimensional and the rules are those of the Game of Life, (9).

Initialisation in $glider(A)$ corresponds to a choice of A in (25) satisfying $\{\ \} \subset A \subseteq N(0,0)$. Evidently that is satisfied by (the characteristic function of) A_0 from (12):

$$init \ := \ x = ((m,n) \in A_0)\,.$$

Next, calculus helps us to establish *glider* thus initialised.

Theorem 4. The specification $glider(A_0)$ is satisfied provided

$$\forall\, t : \mathbb{N} \cdot \forall\, i : [0,4) \cdot \sigma^{4t+i} A_0 \ = \ \tau_{(t,-t)}\, A_i\,, \tag{26}$$

a condition that holds under Rule (9).

Proof. First we calculate sufficiency of (26).

$$\begin{array}{ll}
& \exists\, n : \mathbb{N};\ k, l : \mathbb{Z} \cdot\ A_0 \in h(n,k,l) \\
\Leftarrow & \text{choice of } h,\ k \text{ and } l \text{ to reflect target implementation} \\
& A_0 \in h(4,1,-1) \\
= & \text{Definitions (23, 24)} \\
& \forall\, t : \mathbb{N} \cdot \left(\begin{array}{l} \sigma^{4t} A_0 \ = \ \tau_{(t,-t)} A_0 \\ 4t < u < 4(t+1) \ \Rightarrow \ \sigma^u A_0 \subseteq rect(\sigma^{4t} A_0 \ \cup \ \sigma^{4(t+1)} A_0) \end{array} \right) \\
= & \text{calculus, noting that } (\sigma^{4t} A_0 \ = \ \tau_{(t,-t)} A_0)[0/t] \text{ is } true \\
& \forall\, t : \mathbb{N} \cdot \left(\begin{array}{l} \sigma^{4t} A_0 \ = \ \tau_{(t,-t)} A_0 \\ \forall\, i : \{1,2,3\} \cdot \ \sigma^{4t+i} A_0 \subseteq rect(\sigma^{4t} A_0 \ \cup \ \sigma^{4(t+1)} A_0) \end{array} \right) \\
= & \text{Laws (16) to (19)} \\
& \forall\, t : \mathbb{N} \cdot \left(\begin{array}{l} \sigma^{4t} A_0 \ = \ \tau_{(t,-t)} A_0 \\ \forall\, i : \{1,2,3\} \cdot \ \sigma^{4t+i} A_0 \subseteq rect(\tau_{(t,-t)} A_0 \ \cup \ \tau_{(t+1,-(t+1))} A_0) \end{array} \right) \\
\Leftarrow & \text{calculus, with (22)} \\
& \forall\, t : \mathbb{N} \cdot \left(\begin{array}{l} \sigma^{4t} A_0 \ = \ \tau_{(t,-t)} A_0 \\ \forall\, i : \{1,2,3\} \cdot \ \sigma^{4t+i} A_0 = \ \tau_{(t,-t)} A_i \end{array} \right) \\
= & \text{calculus} \\
& \forall\, t : \mathbb{N} \cdot \forall\, i : [0,4) \cdot \sigma^{4t+i} A_0 \ = \ \tau_{(t,-t)}\, A_i\,.
\end{array}$$

Secondly we show that Condition (26) follows from Rule (9), using induction on t. When $t = 0$, (26) is established by initialisation. Assuming (26) with arbitrary t and i, for the case $t+1$ we have

$$\begin{array}{ll}
& \sigma^{4(t+1)+i} A_0 \\
= & \text{calculus} \\
& \sigma^4 \circ \sigma^{4t+i} A_0 \\
= & \text{induction hypothesis (26)} \\
& \sigma^4 \circ \tau_{(t,-t)} A_i \\
= & \text{Theorem 2 (2)} \\
& \tau_{(t,-t)} \circ \sigma^4 A_i
\end{array}$$

$=$ Laws (16) to (19)

$\tau_{(t,-t)} \circ \tau_{(1,-1)} A_i$

$=$ Theorem 2 (1)

$\tau_{(t+1,-(t+1))} A_i$

as required for (26) with substitution of $t{+}1$ for t. □

Now we use the fact that $glider(A_0)$ is invariant in the loop of *ca* to infer the loop body, using exactly the same method as in Section 4.

$x : [\ glider(A_0), glider(A_0)[t{+}1/t]\ ]$

$\sqsubseteq$ definitions, precondition in postcondition, weaken precondition, strengthen postcondition and Theorem 4 (26)

$x : [\ true, \forall\, i : [0,4) \cdot \sigma^{4(t+1)i} A_0 \ =\ \tau_{(t+1,-t-1)}\ A_i\ ]$

$\sqsubseteq$ timing assumption, corresponding to the 1-dimensional case (8), and (9) and (11)

$x : [\ true, x' = (\nu = 3) \vee (\nu = 2 \ \wedge\ x)\ ]$

$=$ semantics of assignment

$x \ :=\ (\nu = 3) \vee (\nu = 2 \ \wedge\ x)\,.$

6. Related Work

6.1. Definition of emergence

Philosophers have been debating the definition of emergence for well over a century [OW05]. Originally this debate focussed on emergence in the natural world, but in more recent years, with the increasing complexity and interconnectivity of devices, emergence in man-made systems has become a pressing topic. Much of the debate has centered around the *functionalist* claim, *e.g.* [Fod74, Fod97], that emergent properties arise autonomously at the higher levels at which they are observed, versus the *reductionist* position, *e.g.* [Wei95, Wei01], that they are the consequence of, and hence can be reduced to, lower-level properties or laws. In other words, functionalists claim that the emergent behaviour of a system or entity cannot be understood and explained in terms of its components and their interactions. The emergence of consciousness from the workings of the human mind is often used to support this view. Reductionists, on the other hand, argue that all emergent properties of a system or entity, including those of the human mind, *can* ultimately be explained in terms of its components and their interactions.

Anderson [And72], a reductionist, attempts a reconciliation of the differing points of view by distinguishing reductionism from *constructivism*. The latter, with which Anderson disagrees, argues that if we can reduce complex systems to simple laws then we should be able to reconstruct them from those laws. Anderson points out that this is not true in the face of scale and complexity.

This debate on the nature of emergence has led to the definition of notions of *weak* and *strong* emergence by Bedau [Bed97, Bed03]. Strong emergence is the type of emergence of the functionalists, and weak emergence that of the reductionists. Many of the recurring examples of emergence discussed in the Complex Systems literature are in fact weak emergence. These examples include birds flocking, ants foraging for food, and gliders in the Game of Life. Each of these can be understood in terms of its components and component interactions.

To rule out trivial properties as being weakly emergent, Bedau requires that, while weakly emergent properties of a system can be derived from its components and interactions, this derivation can be carried out *only* using simulation.

Since our approach aims to relate emergent properties with component behaviour, we are concerned only with weak emergence. However, we believe Bedau's requirement that only simulation can be used to show such a relation is too strict. Bedau's primary example of weak emergence is the glider pattern of the Game of Life. Does simulation occur in our development? Defining simulation to be the direct application of the system's rules to derive the next system state from the current state, then simulation is used only in part of the proof: for Laws (16) and (17). The other laws are established by geometric reasoning. It is important to

note that even in proving (16) and (17), we have done something more abstract than simulation: for example most cells in the complement of $N(0,0)$ are reasoned about in the same way; and those within it are divided into classes to abbreviate the reasoning (Figure 2).

To rule out trivial properties as being weakly emergent, therefore, we take an alternative view to Bedau. Our view [HLRS08] is that an emergent property is one which cannot be expressed at the level of abstraction of its components considered unilaterally.

6.2. Engineering emergence

The engineering of systems with emergent behaviour is an active area of research in the field of MAS. Jennings *et al.* [JSW98] describe how "one must use a laborious process of experimentation, trial and error" to engineer MAS. The state-of-the-art is systematically to run experiments *via* simulations. Specific approaches have been advocated by Edmonds and Bryson[EB04], Fromm [Fro06] and De Wolf and Holvoet [WH05], among others.

Edmonds and Bryson [EB04] strongly advocate using only experimentation, even to the extent of stating that "we will have to give up the illusion that we can *fully* understand our own code". This is in stark contrast to our approach which provides, through refinement steps, a detailed understanding of the way in which emergence arises.

Other authors realise that experimentation alone is not enough. Fromm [Fro06] suggests combining experimentation with modelling and top-down analysis in an iterative two-way approach. De Wolf and Holvoet [WH05] suggest using experimentation together with the current best-practice in requirements analysis and software design. The experimentation in their case is to provide feedback in an iterative software-development life-cycle.

Another approach for engineering systems with emergent properties is to use evolutionary algorithms (at design time) to evolve programs for the system's agents that result in the required global behaviour [ZW07]. In the Organic Computing community [Wür08], systems are designed to evolve (at run-time) to meet global requirements. The underlying philosophy is summarised by von der Malsburg [vdM08]:

"Systems are becoming too complex to be programmed in detail any longer. The principles with which programmers formulate programs in their head have to be installed in the computer, so that it can program itself such as to conform to abstract, human-defined tasks."

For example, Nafz *et al.* [NOS+09] propose the use of a SAT solver to reconfigure automatically the agents and their roles within a system whenever global properties are not met, *e.g.*, as a result of an agent being disabled. Those approaches provide a level of confidence that the desired global behaviour will be met but, unlike ours, cannot guarantee it.

Zambonelli and Omicini [ZO04] define three scales of observation of multi-agent systems and argue that different engineering techniques, including formal methods, are required at the different scales. At the *micro* scale, which is concerned with the internal details of individual agents, they propose incorporating ideas from Artificial Intelligence into standard software engineering practices, and the use of formal methods to handle complexity. At the *macro* scale, which is concerned with the collective behaviour of a system, they propose the development of a catalogue of reusable global behaviour based on insights from complex natural and physical systems. At the *meso* scale which is concerned with agent interaction, they advocate the use of shared interaction infrastructure (such as a blackboard or tuple space [Omi99]) to limit and coordinate interactions. They advocate that this shared interaction infrastructure be formalised so that certain system properties can be guaranteed (as demonstrated in [OOR04]). They stop short, however, of suggesting that emergent (macro-level) behaviour can be guaranteed based on such a formalisation.

6.3. The case against Formal Methods

Polack and Stepney [PS05] argue that an abstract specification of the movement of a glider cannot be refined to the rules of the Game of Life. 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. As we have shown, this is not the case.

Polack and Stepney also argue that the specification of the glider does not provide a way of finding the starting state of the cellular automaton implementation. As discussed in Section 1, the motivation for the

steps of an incremental refinement (including the derivation of the initial state of the implementation) are not part of the formal process. Hence, we would not expect to be able to do such a derivation by refinement. This would be the case even if our system did not exhibit emergent behaviour.

Polack and Stepney are not the only authors to have claimed that it is not possible to verify emergent behaviour using refinement. In the multi-agent systems field, Zambonelli and Omicini [ZO04] as well as De Wolf and Holvoet [WH05] make similar claims based on arguments of Wegner [Weg97]. Wegner, using Gödel's incompleteness theorem, argues that models of interactive systems are necessarily incomplete and therefore that proofs of the existence of correct behaviour is only sometimes possible, and 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. In such cases, reasonable assumptions are made about the open environments in which the systems operate [HJJ03]. The consequence is that the behaviour of the system is only guaranteed under these assumptions. If the assumptions are indeed reasonable, however, this is rarely a problem. In a similar fashion, we can prove the existence of correct behaviour, and nonexistence of incorrect behaviour, of multi-agent systems under reasonable assumptions (as is demonstrated in [WLNM08] and [HW08]).

Edmonds and Bryson [EB04] 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. They point out that refinement cannot be used to automatically derive implementations, and that refinement proofs cannot, in general, be automated. Neither of those well known facts preclude the use of refinement however. Again the argument applies to systems other than multi-agent systems to which formal methods, including refinement, have been successfully applied.

6.4. The support for Formal Methods

We have previously demonstrated that refinement of emergent properties is possible for both the glider pattern of the Game of Life [BCG82] and for the self-organising behaviour of an algorithm for modular robots [SS09b]. In the former case, a formal proof of correctness was provided, and in the latter a strategy for a formal correctness proof. However, we are not the first to provide such support for the use of formal methods in the face of emergence.

Cucker and Smale [CS07b, CS07a] have provided a mathematical proof for the emergence of bird-flocking behaviour. Their comprehensive analysis considers the motion of birds in three dimensions for both discrete and continuous time.

Winfield *et al.* [WLNM08] provide a formal model of an *ad hoc* wirelessly connected swarm of mobile robots which could be used for validating algorithm correctness. Similarly, Hamann and Wörn [HW08] have provided a formal model of a swarm of mobile robots. Both of these papers compare predictions from the formal model with simulation results to show the accuracy of the model despite assumptions made.

Techniques for formally relating emergent properties to component behaviour have also been proposed. Chen *et al.* [CNC07] provide a calculus of complex events which can be used to relate high-level behaviours to component-level rule executions. Zhu [Zhu05] defines the concept of a *scenario* as a combination of components' behaviours that describe a global property. Relations between scenarios are defined for when a scenario is part of another scenario, or a scenario leads to another via the execution of a component-level rule. This enables proofs that a particular scenario, such as the initial state of a system, will lead to a another representing an emergent behaviour.

Our work differs from those approaches in that we use only standard formal techniques: the refinement calculus, and the modelling of a cellular automaton as an initialised loop, a particularly simple application of standard techniques from the field of Distributed Systems [BKS88].

7. Conclusion

The purpose of this paper has been to study the role of Formal Methods, and incremental refinement in particular, in the engineering of systems exhibiting emergent behaviour. There are various (putative) reasons for which it might previously have been thought impossible to perform incremental refinement of such systems: (a) emergent behaviour is not able to be captured formally; (b) incremental refinement reduces

specified behaviour to that of components and by definition emergent behaviour is not so reducible; (c) incremental refinement does not apply to emergent behaviour because it is in principle different from other kinds of functionality.

Accordingly, as surveyed in the previous section, opinion has varied as to the use of incremental refinement for the class of systems exhibiting emergence. We have established, by considering the popularly representative example of the glider in the Game of Life, that it is indeed possible. The incremental refinement given here depends on capturing the emergent behaviour, in this case by use of global information in the form of global time. Thereafter standard techniques of incremental refinement have sufficed.

Thus, in answer to (a), in this example the emergent behaviour has certainly been able to be captured formally. Can it be captured for an arbitrary information system? At this stage we can merely speculate. But the success of mathematics in describing observed behaviour over the past 3 centuries provides reason for optimism.

The apparent paradox of (b) has been resolved above: component interactions lie at a level intermediate between the (global) specification and the (unilateral) behaviours of component implementations. In our example the components, the cells, have been uniform and simple so that intermediate-level design has not been required. Such designs are however of vital importance in general, and their further study is essential.

In answer to (c), incremental refinement of our examples using the refinement calculus has been entirely routine—even simple. Our treatment has been leisurely largely in order to explore (as summarised in the previous section) to what degree simulation is *required*. Naturally, the more complex the system, the more complex its incremental refinement.

We conclude that the approach taken here makes incremental refinement available for the authentication of multi-agent systems and other systems exhibiting emergence.[8] This is important in view of the subtle behaviour of such systems. To what extent has the Game of Life been deceptively atypical? Firstly, in the uniformity and simplicity of its component cells, requiring no intermediate-level structures. Secondly, in the simplicity of the technique for capturing emergence, when in general (as surveyed in Section 3) hybrid and statistical methods may be required. Thirdly, in the non adaptivity of the system, so both specification and design have been straightforward. Further work in all those directions will be of interest. It seems that the primary task, in more realistic systems, will be in capturing emergence. Adaptive systems are a case in point. How, for example, is a machine-learning system to be specified, whose behaviour depends on its training set?

But perhaps cellular automata are not as simple as they first seem. Gruner [Gru09] 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. Application of our approach to such systems is a possible area of future work.

References

[And72] P.W. Anderson. More is different. *Science*, 177(4047):393–396, 1972.

[BCF10] R. Backhouse, W. Chen, and J.F. Ferreira. The algorithmics of solitaire-like game. In C. Bolduc, J. Desharnais, and B. Ktari, editors, *Mathematics of Program Construction (MPC 2010)*, volume 6120 of *LNCS*, pages 1–18. Springer-Verlag, 2010.

[BCG82] E.R. Berlekamp, J.H. Conway, and R.K. Guy. *Winning Ways for your Mathematical Plays*, volume 2. Academic Press, London, 1982.

[Bed97] M.A. Bedau. Weak emergence. In J. Tomberlin, editor, *Philosophical Perspectives: Mind, Causation and World*, volume 11, pages 375–399. Blackwell Publishers, 1997.

[Bed03] M.A. Bedau. Downward causation and autonomy in weak emergence. *Principia*, 6:5–50, 2003.

[BKS88] R.J.R. Back and R. Kurki-Suonio. Distributed cooperation with action systems. *ACM Transactions on Programming Languages and Systems*, 10(4):513–555, 1988.

[CNC07] C.-C. Chen, S. Nagl, and C. Clack. A calculus for multi-level emergent behaviours in component-based systems and simulations. In *Emergent Properties in Natural and Artificial Complex Systems (EPNACS 2007)*, pages 35–51, 2007.

[CS07a] F. Cucker and S. Smale. Emergent behaviour in flocks. *IEEE Transactions on Automatic Control*, 52(5):852–862, 2007.

[CS07b] F. Cucker and S. Smale. On the mathematics of emergence. *Japanese Journal of Mathematics*, 2:197–227, 2007.

[Dam00] R.I. Damper. Emergence and levels of description. *International Journal of Systems Science*, 31(7):811–818, 2000.

[8] Our approach builds on ideas proposed in [HLRS08], elaborated in [SS08] and exemplified in [SS09b]. An early version, [SS09a], was restricted to consideration of just the Game of Life, did not provide incremental refinements and used a different 1-dimensional example.

[EB04] 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)*, pages 938–945. IEEE Computer Society, 2004.

[Fod74] J.A. Fodor. Special sciences (or the disunity of science as a working hypothesis). *Syntheses*, 28:97–115, 1974.

[Fod97] J.A. Fodor. Special sciences; still autonomous after all these years. In J. Tomberlin, editor, *Philosophical Perspectives: Mind, Causation and World*, volume 11, pages 149–163. Blackwell Publishers, 1997.

[Fro06] J. Fromm. On engineering and emergence, 2006. arXiv preprint `http://arxiv.org/abs/nlin.AO/0601002`.

[Gar70] M. Gardner. Mathematical games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, pages 120–123, 1970.

[Gru09] S. Gruner. Mobile agent systems and cellular automata. *Autonomous Agents and Multi-Agent Systems*, 20(2):198–233, 2009.

[HJJ03] I.J. Hayes, M. Jackson, and C.B. Jones. Determining the specification of a control system from that of its environment. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *Formal Methods Europe (FME 2003)*, volume 2805 of *LNCS*, pages 154–169. Springer-Verlag, 2003.

[HLRS08] J. Hu, Z. Liu, G.M. Reed, and J.W. Sanders. Ensemble engineering and emergence. In M. Wirsing, J.-P. Banâtre, M. Hölzl, and A. Rauschmayer, editors, *Software-Intensive Systems and New Computing Paradigms: Challenges and Visions*, volume 5380 of *LNCS*, pages 162–178. Springer-Verlag, 2008.

[HW08] H. Hamann and H. Wörn. A framework of space-time continuous models for algorithm design in swarm robotics. *Swarm Intelligence*, 2:209–239, 2008.

[JSW98] N.R. Jennings, K. Sycara, and M.A. Wooldridge. Roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–23, 1998.

[Mor94] C.C. Morgan. *Programming from Specifications*. Prentice-Hall International, second edition, 1994.

[NOS+09] F. Nafz, F. Ortmeier, H. Seebach, J.-P. Steghöfer, and W. Reif. A universal self-organisation mechanism for role-based Organic Computing 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*, pages 17–31. Springer-Verlag, 2009.

[Omi99] A. Omicini. On the semantics of tuple-based coordination models. In *1999 ACM Symposium on Applied Computing*, pages 175–182. ACM Press, 1999.

[OOR04] A. Omicini, S. Ossowski, and A. Ricci. Coordination infrastructures in the engineering of multiagent systems. In F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*. Kluwer, 2004.

[OW05] T. O'Connor and H.Y. Wong. Emergent properties. In E.N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Stanford University, 2005.

[PS05] F. Polack and S. Stepney. Emergent properties do not refine. In J. Derrick and E. Boiten, editors, *International Refinement Workshop (Refine'05)*, volume 137, Issue 2 of *Electronic Notes in Theoretical Computer Science*, pages 163–181. Elsevier, 2005.

[SS08] J.W. Sanders and G. Smith. Formal ensemble engineering. In M. Wirsing, J.-P. Banâtre, M. Hölzl, and A. Rauschmayer, editors, *Software-Intensive Systems and New Computing Paradigms: Challenges and Visions*, volume 5380 of *LNCS*, pages 132–138. Springer-Verlag, 2008.

[SS09a] J.W. Sanders and G. Smith. Refining emergent properties. In E. Boiten, J. Derrick, and S. Reeves, editors, *International Refinement Workshop (Refine 2009)*, volume 259 of *Electronic Notes in Theoretical Computer Science*, pages 207–233. Elsevier, 2009.

[SS09b] 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*, pages 90–104. Springer-Verlag, 2009.

[vdM08] C. von der Malsburg. The organic future of information technology. In R.P. Würtz, editor, *Organic Computing: Understanding Complex Systems*, pages 7–24. Springer-Verlag, 2008.

[WBHR08] M. Wirsing, J.-P. Banâtre, M. Hölzl, and A. Rauschmayer, editors. *Software-Intensive Systems and New Computing Paradigms: Challenges and Visions*, volume 5380 of *LNCS*. Springer-Verlag, 2008.

[Weg97] P. Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):80–91, 1997.

[Wei95] S. Weinberg. Reductionism redux. *The New York Review of Books*, 1995.

[Wei01] S. Weinberg. *Facing Up*. Harvard University Press, 2001.

[WH05] 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.

[WLNM08] A. Winfield, W. Liu, J. Nembrini, and A. Martinol. Modelling a wireless connected swarm of mobile robots. *Swarm Intelligence*, 2:241–266, 2008.

[Wür08] R.P. Würtz, editor. *Organic Computing: Understanding Complex Systems*. Springer-Verlag, 2008.

[Zhu05] H. Zhu. Formal reasoning about emergent behaviour in MAS. In *International Conference on Software Engineering and Knowledge Engineering (SEKE'05)*, 2005.

[ZO04] 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.

[ZW07] M. Zapf and T. Weise. Offline emergence engineering for agent societies. In *European Workshop on Multi-Agent Systems (EUMAS'07)*, 2007.

Appendix

Section 2 contains a sketch of how the refinement calculus may be used to provide an incremental refinement for array summation. Here the ideas are combined to given an actual incremental refinement.

$s : [true, s' = \sum_{0 \le j < n} a[j]]$

$\sqsubseteq$ refinement calculus: introduction of local block

var $i : \mathbb{N} \cdot$
  $s, i : [true, s' = \sum_{0 \le j < n} a[j]]$
rav

$\sqsubseteq$ refinement calculus: introduction of a loop

var $i : \mathbb{N} \cdot$
  $i, s := 0, 0$ ⨾
  do $i < N \ \rightarrow$
    $i, s : [inv \wedge i < N, inv']$
  od
rav

$\sqsubseteq$ refinement calculus: strengthen postcondition

var $i : \mathbb{N} \cdot$
  $i, s := 0, 0$ ⨾
  do $i < N \ \rightarrow$
    $i, s : [inv \wedge i < N, inv' \wedge i' = i + 1]$
  od
rav

$\sqsubseteq$ predicate calculation

var $i : \mathbb{N} \cdot$
  $i, s := 0, 0$ ⨾
  do $i < N \ \rightarrow$
    $i, s := i + 1, s + a[i]$
  od
rav