Emergence, Abstract Multiagent Systems and Multi-state Networks

James Adam, University of Essex, Wivenhoe Park, Colchester, UK CO4 3SQ
[jgadam@essex.ac.uk]

Abstract
Multiagent systems and multi-state networks are two distinct systems in which emergent phenomena are often cited. An abstract specification for simple multiagent systems is presented, and equivalence between multiagent systems using this specification and multi-state networks (a generalisation of systems including Cellular Automata) is demonstrated. The limitations of this translation are highlighted, and similarities and differences in the phenomenon of emergence in the two systems are discussed.

Introduction
Emergence is a term used to describe the production of certain high-level effects of behaviours from a system whose components are simple, often homogenous, and interact simply and locally. Emergence and emergent phenomena have been studied extensively in the context of Boolean and multi-state networks [Kauffman 1991, Wuensche 1992 & 1998, Solé 2000] in their many guises, such as cellular automata [Neumann 1966, Wolfram 1984]. Various properties of these networks (such as the degree of node connectivity) have been shown to influence the abilities of such networks to produce emergent, self-organizing structures and phenomena ('gliders' being an often-quoted example).

Emergence is also commonly cited as an effect in multiagent systems, such as the production of coherent nest structures [Theraulaz 1995], or successful cooperation in gathering and sorting tasks [Holland 1999, Steels 1989, Drogoul 1994]. Emergence is also used to describe 'social' behaviours within such systems, such as group movement (flocking, aggregation, dispersal [Reynolds 1987, Mataric 1992]) and more intelligent social behaviours such as team formation [Takadama et al. 1997] and the construction of social hierarchies [Doran 1995]. This investigation has begun to look at a possible correspondence between emergent behaviour in multiagent systems and emergent phenomena in multi-state networks.

If such a link exists, it may give more weight to the idea of 'strong emergence' [Bedau 1997], or the existence of a process or phenomenon which exists beyond notions of emergence which rely on the cognitive capabilities of observers [Ronald 1999, Darley 1994]. The demonstration of such a link may also give insights as to the nature of emergent processes within both of these currently separate areas of research.

Multi-state Networks
Multi-state networks (MSNs) are a generalization of Boolean networks [Kauffman 1991, Wolfram 1984, Solé 2000]. An MSN consists of a fixed number of nodes, each capable of being in one of a number of 'states'. In Boolean networks, only two possible states are allowed – on, or off (1 or 0), but in an MSN, the number of possible states can be more than two. Each of these nodes is connected to several other nodes, forming the 'neighbourhood' of the original node.

Each node is assigned an initial state, taken from the global set of possible states. At each time step (t), the system is advanced by computing each node's next state using the states of the nodes in its neighbourhood the previous time step (t-1). The function used to compute each node's state transition may be identical for all nodes within the network, or may be unique to that node, and normally takes the form of a Boolean function or a lookup table.

A well-known example of a Boolean network is Conway's 'Game of Life' and other related Cellular Automata (CA). In the classic Game of Life CA, nodes (cells) are arranged in a 2-dimensional grid, and connected to the eight surrounding cells. The possible states a cell can be in are 'Alive' and 'Dead'. The function used to determine a cells state at the next timestep (t+1) considers the number of 'Alive' cells within the cell's neighbourhood:

- If less than 2 cells are Alive, then be Dead (loneliness)
- If 2 or 3 cells are Alive, then be Alive
- If more than 3 cells are Alive then be Dead (overcrowding)

This extremely simple system gives rise to many complex patterns and 'emergent' phenomena, including 'gliders' 'glider-guns'

and the possibility of universal computation [Neumann 1966, Wolfram 1984].

Multiagent Systems

A Multiagent System (MAS) is simply a collection of agents which can interact, possibly within an external environment. The idea of an 'agent' is itself only loosely defined – it is most often an entity which autonomously exhibits behaviours, which in turn affect the state and behaviours of the other agents within the system. If the system is situated within an environment, the agents can typically modify and sense changes in their local environment.

An Abstracted Multiagent System

An abstraction which can be used to describe simple multiagent systems – AbMAS – has been developed as follows: A multiagent system M consists of i agents $A = \{a_1, a_2, a_3, \ldots, a_i\}$. Each agent a_x consists of a working memory WM_x and an ordered list of n rules $R_x = \{r_{x1}, r_{x2}, r_{x3}, \ldots, r_{xn}\}$. A working memory is simply a (possibly empty) set of symbols taken from a global alphabet S. A rule r_{xk} is simply a set of conditions C_{xk} and assertions A_{xk}. Each condition is a symbol taken from S. Each assertion is of the form (j, k, *add*), where j is a symbol from S, k is the reference of another agent within the system ($1 \leq k \leq i$) and *add* is a Boolean flag with values of either *true* or *false*.

The conditions of each rule specify the state an agent must be in before the rule could be fired. The assertions within a rule represent the messages or actions an agent will send if that rule fires. If the *add* flag is set to 'true', the element specified will be added to the working memory of the agent corresponding to the agent identifier specified. If the *add* flag is false, the element will be removed.

For example, consider the following rule:

```
Rule(
     {'a', 'b'},
     {('x', 2, true), ('b', 1,
     false)}
)
```

This rule requires the elements 'a' and 'b' to be present in the agent's WM. If the rule is selected for firing (see below), the element 'x' will be added to the WM of agent 2, and the element 'b' will be removed from agent 1's WM. It is important to note that elements may only be removed from an agent's working memory by rules from that agent itself. The above rule therefore can only belong to agent 1. This constraint reflects the nature of individuality within agent-based systems – agents do not have access to the internal state of other agents.

During execution, each cycle consists of two stages – rule selection and rule firing. During the rule selection stage, each agent selects a rule to fire by finding the first rule within the ordered set of rules whose condition set elements are all found within the current contents of that agent's WM. The ordering imposed on the rule set is as follows:

- more specific rules (i.e. rules with a larger set of conditions) are ordered before less specific rules
- rules with equal specificity (i.e. an equal number of conditions) are ordered arbitrarily by the designer of the agent.

Once each agent has selected a rule, the simulation execution enters the rule firing stage, during which the effects of the assertions are implemented. The rule firing stage is again split into two stages -

For each agent in the system:

1. All assertions which refer to this agent are processed, for each agent within the system (updating the agent's internal state) in the order they are specified within the rule.
2. All other assertions (symbols to add to other agents – 'messages') are then processed for every agent within the system.

The separation of the two sub-stages is required to avoid ambiguity in the outcome of rules firing simultaneously within the MAS. For example, if one agent were to add an element to a WM while another agent removes the same element from the same WM, the outcome is unclear. However, because only the rules within each agent can remove elements from that agent's WM, all element removal assertions are processed during the first sub-stage, removing the conflict.

Mapping a MAS to an MSN

It is now relatively easy to transform any multiagent system adhering to the above specification into a multi-state network. Each agent in the MAS is represented within the network as a single node. The lookup tables for the MSN are generated by running the simulation for all possible combinations of agent WM contents, each for a single cycle. A large table is constructed with two main columns – the system state before the cycle is run (MAS(t)), and the state after the cycle has finished (MAS(t+1)). Both of these columns are further divided into sub-columns for each

agent's WM. In this manner, each table entry holds the state of each agent (represented by its WM) in the system both before and after a simulation cycle. The individual lookup tables for each node can be extracted from this global table with minimum effort. To extract the lookup table for $node_i$ (the node which corresponds to $agent_i$), simply create a copy of this large table, but remove all sub-columns from the MAS(t+1) column except that representing the WM of $agent_i$.

Two further operations can be performed to reduce this table in size. To remove redundant rows, compare the state of $node_i$ in the MAS(t) column with the state in MAS(t+1) – if there is no change, this row can be discarded. Secondly, to reduce the size of $node_i$'s neighbourhood (which will currently be the entirety of the MSN), the following algorithm can be used:

1. For each node n in the MAS($_t$) column
   a. For each combination of the states of the other nodes (i.e. all nodes EXCEPT $node_i$)
      i. For each state of $node_n$, if the state of $node_i$ under MAS(t+1) differs from the state of $node_i$ under MAS(t), set a flag to true.
2. If the flag is still false, $node_n$ never has any influence on the state of $node_i$, so:
   a. The sub-column of MAS(t) in the table for $node_i$ which refers to the state of $node_n$ can be removed.
   b. Duplicate rows within the table will now exist after the column removal – these can also be thrown away.

The final neighbourhood of $node_i$ is all of the nodes which have sub-columns within the MAS(t) column of this table. The possible states of a node (S') are all possible subsets (including the empty set) of the alphabet S.

From MSN back to a MAS

A significant aim of this investigation has been to developing the process to extract a corresponding MAS from an MSN. A simple approach to achieving the reverse mapping is based on a simple reversal of the manipulation of the lookup tables generated above.

In order to determine the effects of each agent on all other agents – the rules of each agent – the individual lookup tables of each node must be recombined into an amalgamated table, as generated in the previous section. This can be achieved as by effectively reversing the process used above. First, the neighbourhoods of each node must be extended to include the entire network again. This can be achieved by recreating the duplicate rows within the table (step 2.b above) – each row must be duplicated for all possible combinations of states of the nodes which are not already represented in the MAS(t) column.

Next, the states of nodes must be converted into WM contents – this is done by creating a new alphabet T, whose size is given by $f(|S|) = x$ where $2^{x-1} < |S|$ and $2^x > |S|$. T should then be ordered. Each state in S' can then be given a binary index, e.g. 'a' = 00, 'b' = 01, 'c' = 10 and 'd' = 11. Finally, each digit within the binary index should be considered to indicate the presence (in the case of a 1) or absence (in the case of 0) of the corresponding element of T (according to the order of the elements of T). For instance, if 'A' is first within T, the presence of a 1 in the least significant digit of the binary index of the state means that 'A' is present in the WM contents. Thus, the state 'd', with the binary index 11, is converted into the WM contents {'A','B'}. In the case of index 0, the WM of the agent is 'empty'.

To extract the rules themselves, the following procedure is used:

1. Generate the MAS alphabet T, as above and substitute all node states with the equivalent symbols.
2. For each $node_i$ in the network
   a. Create a new $agent_i$
   b. For each row where all nodes except $node_i$ have an 'empty' WM (rules may only refer to the agents own WM, so the states of other nodes can't be used as rule conditions and can be ignored):
      i. Compare the state of each node at MAS(t) and MAS(t+1), noting differences.
      ii. Create a new rule within $agent_i$ where the conditions are the state of $node_i$, and the assertions are the differences determined in the previous step (e.g. if node3 has state 'A' at time (t) and the symbols 'B' and 'C' at time (t+1), add assertions ('A', 3, false), ('B', 3, true) and ('C', 3, true) to the new rule).

This process will extract a MAS which is *equivalent* to the original MAS – that is, a set of rules which will produce an identical *table* when converted to an MSN as the set of rules from the original MAS. Because the table represents the behaviour of the system in every possible state, *equivalent* systems which are in identical states at time t, will also be in identical states at time t+1.

It is worth noting that the *rules* generated by this reverse process may not be identical to the original MAS rules. This does not mean that the MAS extracted is not equivalent – in fact this process could be used as a means to identify groups of equivalent MAS, which are functionally identical (i.e. they perform the same transitions in identical system states).

However, there are significant problems with this approach. The size of the generated table can be quite large (the number of rows = $2^{n\,|S|}$, where n is the number of agents in the system, and |S| is the number of symbols in the alphabet). Consider a MAS with 10 agents, and an alphabet of 10 symbols. The size of the table generated would be 2^{100}, or 1,267,650,600,228,229,401,496,703,205,376 rows. Clearly this size of table is infeasible to manipulate – in the worst case the actual neighbourhoods of each node would be every other node, and no further size reductions could be made, resulting in 10 node tables each of this size. It is clear that for translation from MAS to MSN and back to be practical, an alternative method must be found.

Rule Manipulation Method

As an alternative to generating the node lookup tables using 'brute force' (by running the simulation using every possible system state), an approach which works directly on the rules within the system has been developed. This method significantly reduces the amount of intermediate information which must be used to extract node rules. Using the following example MAS -

```
agent 0 {
  rule0.1: a,b → (a,0) & (b,1)
        ('a' & 'b' are preconditions;
        the assertions are 'a' added to
        agent 0, and 'b' to agent 1)
  rule0.2: c,b → (c,0) & (c,2)
  rule0.3: a → (a,1) & (a,2)
}

agent 1 {
  rule1.1: a → (b,1)
  rule1.2: b → (a,1) & (b,0) & (c,2)
}

agent 2 {
  rule2.1: a,c → (b,0) & (b,1) & (b,2)
  rule2.2: c → (a,2)
}
```

- to extract the MSN from a MAS using rule-manipulation, the following procedure is used:

1. For every $agent_i$ in the system:
   a. Label each rule using an ordered sequence of labels matching the order of the rules within that agent, e.g. $rule_{i.1}$, $rule_{i.2}$, etc
   b. Create a node ($node_j$) for $agent_i$
2. For every $agent_i$ in the system:
   a. For each $rule_{ix}$:
      i. Extract and label the sub-rules from $rule_{ix}$ (see below)
      ii. Add each $sub\text{-}rule_{ixy}$ to the appropriate node, i.e. insert $sub\text{-}rule_{abx}$ into $node_x$ and $sub\text{-}rule_{aby}$ into $node_y$
3. For each $node_j$:
   a. 'Multiply' the node's sub-rules (see below)

Details:

2.a.i – Extract and label the sub-rules from $rule_{ix}$

1. Transform the conditions of the rule by attaching the agent's label *I* to each condition, i.e.

   $rule_{0.1}$ a, b → …

   becomes

   $rule_{0.1}$ (a,0),(b,0)→ …

2. Create sub-rules using all the conditions of $rule_{ix}$ and only the assertions targeting a single $agent_a$, i.e. from

   $rule_{0.1}$ (a,0),(b,0)→(a,0)&(b,1)

   create two sub-rules:

   $sub\text{-}rule_{0.1.0}$ (a,0),(b,0)→(a, 0)

   $sub\text{-}rule_{0.1.1}$ (a,0),(b,0)→(b, 1)

3.a 'Multiply' the sub-rules

For each id *i* in $sub\text{-}rules_{i.j.k}$ (i.e. for each agent which contributed sub-rules to this node), create an ordered set and add all $sub\text{-}rules_{i.j.k}$ to that set (in numerical order over *i*). For instance, considering Node (0) in the example, the sets would be:

[0.1, 0.2], [1.2], and [2.1]

where 0.1…2.1 are sub-rules, and 0.1 is 'above' 0.2.

Rules are then created as follows. Maintaining the order of the sub-rules within the sets, create all rules which consist of a sub-rule from all *i* sets. In our example, first rule to be created will be

```
0.1 + 1.2 + 2.1
=  (a,0),(b,0)→(a,0)
 + (b,1)→(b,0)
 + (a,2),(c,2)→(b,0)
= (a,0),(b,0),(b,1),(a,2),(c,2) →
(a,0) & (b,0) & (b,0)
```

Note that the duplicate (b,0) can be removed. The next rule to be created will be:

0.2 + 1.2 + 2.1.

This is continued until all combinations of sub-rules, where a single sub-rule is drawn from all *i* sets, have been created. Next, rules are created where a single sub-rule is drawn from

all sets except one. The sub-rules created will be:

```
0.1 + 1.2
0.2 + 1.2
0.1 + 2.1
0.2 + 2.1
1.2 + 2.1
```

This generation of rules from progressively fewer sets continues, until finally rules from sub-rules from only a single set have been created (these final rules are, in fact, the sub-rules themselves). The final set of rules created for Node 0 in our example is (in order):

```
0.1 + 1.2 + 2.1
0.2 + 1.2 + 2.1
0.1 + 1.2
0.2 + 1.2
0.1 + 2.1
0.2 + 2.1
1.2 + 2.1
0.1
0.2
1.2
2.1
```

Applying Rule-splitting to MAS→MSN

The essence of the rule-splitting procedure to translate Multiagent Systems into Multi-state Networks is as follows:

- Split agent rules into sub-rules containing assertions to single WMs
- Create nodes with all system sub-rules which assert to a single WM
- Combine all sub-rules within each node

In this case, each of the conditions for a given rule remains together in all of the sub-rules created. If the set of conditions originating from a rule matches part of the current state of the system, then all sub-rules derived from that rule (or at least one rule which was 'multiplied' from those sub-rules) will fire, and all the original assertions will be applied.

However, to apply this type of rule manipulation in order to perform the reverse translation, the following procedure is required:

- Split node rules into sub-rules containing *conditions* from single WMs (because conditions within agent rules can only refer to a single WM)
- Combine all sub-rules within the system whose conditions refer to the same WM and use them to reconstruct the agent.

This reversal is not possible, because after such a process, the conditions of a single node rule are split between a number of agent rules. It is now impossible to determine whether any of the assertions will be made because rules cannot access the entire condition set.

Consider two rules:

1. (a, 0) → (x, 2) [from agent '0']
2. (b, 1) → (y, 2) [from agent '1']

Once this MAS is translated into an MSN, the following rule will exist:

(1 & 2). (a,0), (b,1) → (x,2) & (y,2) [from node '2']

Now consider translating the same rule from an MSN node rule to a MAS agent rule. Clearly the rule must be split into two sub-rules – one for agent '0' and another for agent '1'. But how can we separate the assertions? The rule:

(a,0), (b,1) → (x,2) & (y,2)

becomes two sub-rules:

i) (a,0) → x / y / x & y ?
ii) (b,1) → x / y / x & y ?

There is no way of appropriately apportioning the assertions between these rules that will guarantee that the system's *equivalence* is maintained. In fact, there is no guarantee that both of these sub-rules will be selected to fire during the same system cycle, and if either sub-rule carries all of the assertions of the node rule, then the other conditions the node rule contained become irrelevant. While it is obvious in this case that a solution does exist, it is trivial to create an MSN with rules which cannot be divided into sub-rules and thus translated into agent rules; the translation from any arbitrary MSN to a corresponding MAS can not be guaranteed to succeed.

A General Rule System

As can be seen from above, it is clear that general MSN→MAS translation is not possible. This is due to the fundamentally different natures of the interactions between active entities (agents and cells/nodes) within the two different systems. Specifically, the problem lies within the different restrictions on rules in the two systems.

In order to clarify this idea, we can consider both MASs and MSNs to be specialisations of

a 'General Rule System' (GRS). A GRS consists of:

- a set of Working Memories (WMs), which can each store a number of different Symbols
- a set of Rules or the form *<condition 1> & <condition 2> & ... -> <assertion 1> & <assertion 2> & ...* where each condition and assertion contains a Symbol and a reference to a specific Working Memory. In addition to this, each assertion also has a flag which have two values - 'add' or 'remove'.

A condition is satisfied when the WM it refers to contains the symbol associated with that condition. An assertion is applied by adding or removing (as determined by the flag) the symbol associated with the assertion from the WM specified. For a rule to be eligible for 'firing', all conditions must be satisfied. When a rule 'fires', its assertions are applied. In this way, as the system 'runs', rules fire only when they are eligible (particular strategies for selecting which rule(s) to fire can be devised as appropriate to the application of the system).

A MAS imposes the following constraint on the GRS: *All conditions within a single rule must have the same Working Memory reference*. This reflects an agent's ability to make decisions based only on internally available information, whilst being able to send 'messages' to other agents through various forms of communications.

An MSN imposes a different constraint on the GRS: *All assertions within a single rule must have the same Working Memory reference.* This mirrors the function of a node's lookup table, which determines a node's next state based on the current states of other nodes around it (its neighbourhood). In both cases, at each system time step only a single rule (if any) from each agent/node is fired.

The conversion from MAS to MSN is possible because a MAS rule can be split into a number of rules, each containing all the conditions and only the assertions relating to one WM, thus giving a valid MSN rule. Because all the rule's conditions refer to the same WM, when one of the sub-rules is eligible for firing, the rest are also eligible. All sub-rules can therefore fire, applying all the original assertions within the same time step. However to move from a MSN to a MAS requires that the conditions from a single MSN rule be split between a number of MAS rules. It is not, in general, possible to split the rule in this way, because only some of the conditions from the original rule need to be satisfied for one rule fragment to fire, and there is no way to guarantee that all fragments of a given MSN rule will fire simultaneously at any time (which is what is required to give *equivalence*, i.e. the same behaviour as the original MSN).

In conclusion, while it is possible within the AbMAS specification to translate any appropriately specified MAS into an MSN, and the MSN produced back into a MAS, it is not possible to translate any arbitrary MSN into a corresponding MAS.

Emergence

The primary motivation for this investigation was to attempt to identify any common features of the emergence demonstrated within the Multiagent Systems field and that cited in the Multi-state Network / Cellular Automata fields of research.

Emergence within MSNs

Emergence within MSNs is exemplified by the 'glider' example – a configuration of cells which moves though the space of the CA. However it is misleading to believe that the glider is anything other than a pattern which moves around the cells. The rules of the Game of Life can be interpreted as a density distribution mechanism: if a large group of cells (i.e. 3 or more) are currently 'alive', cells within the centre of that group are likely to 'die', and cells nearby the densest part of the configuration are likely to become 'alive' during the next time step.

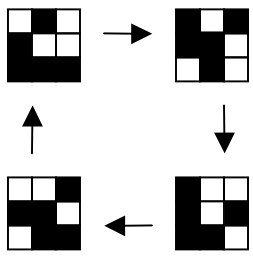

Figure 1. The configurations of a 'glider' as it moves through 'space'.

The glider pattern is clearly densest at one corner, and this is the direction in which the pattern will move. So it can be said that the rules of the Game of Life support this pattern – because it maintains an imbalance of density, it is weighted in one direction and will reproduce itself as it 'moves', and continue being reproduced until the pattern is disrupted (a 'collision').

All emergent phenomena within cellular automata, and multi-state networks in general, take the form of patterns of the states of cells throughout the execution of the system. Some patterns appear as (eventually) static structures such as the 'highway' produced by the Langton Ant [Cohen 1994] or the patterns produced by sorting CAs [Mitchell et al. 1994, Nishio 1981], while others are locally related configurations which appear to move around the 'world' and interact with other patterns, such as the numerous objects found in the 'Life' CA (see [Weisstein 2002] for an excellent catalogue).

Emergence within MASs

Emergence as demonstrated in multiagent systems takes many different forms:

- the appearance of structures within the environment (nests, construction of paths)
- the production of some 'physical' arrangement of the agents themselves (flocking, herds)
- patterns in the behaviour of agents (formation of teams or adoption of social roles)
- patterns in the behaviour of the whole system.

An emergent behaviour of a multiagent system is some behaviour of the system as a whole which has some identifiable effect on the world, or even the system itself (second order emergence [Gilbert 1994]). It is easy to describe and identify the behaviours of individual agents, because they are typically formulated, specified and described in such terms. But how do you identify the behaviour of a whole system? Individual behaviour can be broken down into repeated sequences of atomic actions, but the non-linear combination of many such actions does not readily lend itself to decomposition.

Instead, behaviour must be considered as a pattern in the measurements of some aspect of the system. In the case of a wall-following robot [Steels 1994], while it can be described as performing two distinct 'behaviours' (moving towards the wall and keeping a given distance from the wall), in fact to an observer the overall behaviour of the robot is described by the distance between the robot and the wall. For gathering or sorting multiagent systems [Holland 2000, Steels 1989, Drogoul 1994], the emergence is shown through a pattern in the measurement of the maximum distance between any two items. The choice of measurement is made by the designer or observer, based on their intuitive understanding of the system, and this choice is crucial in determining whether or not any emergence is identified. For instance, if the measurement chosen for Steels' rock-gathering simulation [Steels 1989] was the average distance between agents, instead of the number of rocks gathered, no emergent functionality would be identified.

AbMAS does not incorporate any contextual information to the observer and as a result of this the choice of measurements and identification of patterns within it is arbitrary. It even lacks the spatial notion present in cellular automata (a 2D grid in the case of the Game of Life), which certainly helps identify gliders and other such patterns within the system. However it may be true that other patterns exist within such systems, which are hidden or obscured because they do not correspond to the spatial representation within the system. For example, much of the information present in a computerised image is as a result of the arrangement of the pixels; if the pixels are moved around randomly, while the states of the image's individual components have not changed, it will no longer be clear what the image represented. There are very likely to be many patterns of symbols present within a system's Working Memories or node states, and only a small few of these patterns will correspond to the 'emergence' identified in the original system. If there is no qualitative difference between these 'emergent' and 'non-emergent' patterns, then the identification of any emergent aspects of the system must require access to the specific context that the system was originally presented in; the measurements and identification of patterns must be done by some observer.

Summary

A framework for specifying and abstracting multiagent systems has been presented, and a correspondence and translation between multiagent systems within this framework and multi-state networks shown. The limitations of this translation have been demonstrated and the source of this limitation – incompatible differences between the constraints on the rules within the two types of system – has also been identified. The notion of 'emergence' when considered using these systems must be reduced to patterns within the states of elements within the system. The identification of suitable patterns has great bearing over the attribution of 'emergent' phenomena to the

system. In such an abstract framework and with a complex system many patterns will exist, and it is unclear if those corresponding to 'emergent' phenomena within the original system can be distinguished from other patterns which had not been associated with any emergence previously. This may lend weight to the notion of emergence as a label which requires an observer to bestow it upon a system.

References

[Bedau 1997] Bedau M. A., "Weak Emergence". In J. Tomberlin (ed.), Philosophical Perspectives: Mind, Causation, and World, Vol. 11 (Malden, MA: Blackwell), pp. 375-399. 1997.

[Cohen 1994] Cohen, J. & Stewart, I., "The Collapse of Chaos; simple laws in a complex world". Penguin, Viking, New York. 1994.

[Darley 1994] Darley V., "Emergent phenomena and complexity". In R. Brooks & P. Maes (eds.), Artificial Life IV, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, MIT Press. 1994.

[Doran 1995] – Doran, J., and Palmer, M., "The EOS project: integrating two models of Palaeolithic social change". In N. Gilbert and R. Conte (eds.) Artificial Societies. UCL Press. 1995.

[Drogoul 1994] Drogoul, A., and Ferber, J., "From Tom Thumb to the Dockers: Some Experiments with Foraging Robots". In J.-A. Meyer, H. Roitblat and E. Werner (eds.), From Animals to Animats: Second Conference on Simulation of Adaptive Behavior (SAB 92). MIT Press. 1994.

[Gilbert 1994] Gilbert N., "Emergence in social simulation". In N. Gilbert and R. Conte (eds.), Artificial Societies: The computer simulation of social life, pp. 144-156. UCL Press, London. 1994.

[Holland 1999] Holland, O. and Melhuish, C., "Stigmery, Self-Organisation and Sorting in Collective Robotics". Artificial Life, 5, pp. 173-202. 1999.

[Kauffman 1991] Kauffman, S., "The Origins of Order: Self-Organization and Selection in Evolution." Oxford University Press, Oxford, 1991.

[Mataric 1992] Mataric M., "Designing emergent behaviours: from local interactions to collective intelligence". In Second International Conference on Simulation of Adaptive Behaviour, pp. 432-441. 1992.

[Mitchell et al. 1994] Mitchell, M., Crutchfield, J. P., and Hraber, P. T., "Evolving Cellular Automata to perform Computations: Mechanisms and Impediments". Physica D, 75, pp. 361-391. 1994.

[Neumann 1966] Neumann von, J. "Theory of Self-Reproducing Automata", University of Illionois Press, Champain, IL. 1966.

[Nishio 1981] Nishio, H., "Real time sorting of binary numbers by 1-dimensional cellular automata,". Kyoto University report. 1981.

[Reynolds 1987] Reynolds, C. W., "Flocks, Herds and Schools: A distributed behavioural model.", In Computer Graphics, 21, pp. 25-34.

[Ronald 1999] Ronald, E. M. A., Sipper, M., and Capcarrere, M. S., "Design, Observation, Surprise! A Test for Emergence", Artificial Life, 5, pp. 225-239. 1999.

[Steels 1989] Steels L. "Cooperation between distributed agents through self-organisation". In Yves Demazeau and Jean-Pierre Muller (eds.), Decentralised A.I., pp. 175-196, North-Holland, 1990.

[Solé 2000] Solé, R. V., Luque, B., and Kauffman, S., "Phase Transitions in Random Networks with Multiple States". Santa Fe Working Paper 00-02-011. 2000.

[Steels 1994] Steels L. "The Artificial Life roots of Artificial Intelligence". Artificial Life, vol. 1, pp. 75—110. 1994.

[Takadama et al. 1997] Takadama K., Hajiri K., Nomura T., Nakasuka S., and Shimohara K. "Learning Model for Organizational Learning in Coexistent Sub-Groups of Swarm Robots". In The 4th European Conference on Artificial Life (ECAL97), 1997.

[Theraulaz 1995] Theraulaz, G., and Bonabeau, E., "Coordination in distributed building". Science, 269, 686-688. 1995.

[Wavish 1992] Wavish P., "Exploiting emergence behaviour in multi-agent systems". In E. Werner and Y. Demazeau (eds.), Decentralized A.I. 3 – Proceedings of the Third European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW-91), pp. 297-310. 1992.

[Weisstein 2002] URL: http://www.ericweisstein.com/encyclopedias/life/

[Wolfram 1984] Wolfram S., "Universality and Complexity in Cellular Automata", Physica 10D, 1-35. 1984.

[Wuensche 1992] Wuensche, A., and Lesser, M., "The Global Dynamics of Cellular Automata." Santa Fe Institute Studies in the Sciences of Complexity. Reference Volume I. Reading, MA: Addison-Wesley. 1992.