Formal Verification of Emergent Properties

Kamal Boumaza
LISCO: Laboratory of complex systems engineering, Computer science department
Badji Mokhtar University, Annaba, B.P. 12, 23000, Algeria
E-mail: kamel.boumaza23@gmail.com

Cherif Tolba
Computer science department, Badji Mokhtar University, Annaba, B.P. 12, 23000, Algeria
E-mail: cherif.tolba@univ-annaba.dz

Iulian Ober
ISAE-Supaero, University of Toulouse, France
E-mail: iulian.ober@isae-supaero.fr

Keywords: formal verification, emergence, model checking, refinement, game of life, boids

Received: Mai 11, 2020

Complex systems and systems of systems (SoS) are systems characterized by the interconnection of a large number of components or sub-systems. The complexity of such systems increases with the number of components and their manner of connectivity. The global behaviour of complex systems and SoS exhibits some properties that cannot be predicted by analysing components or sub-systems in isolation. The verification of these properties called "emergent properties" is considered a crucial issue when engineering such systems. The purpose of this paper is to give an overview and verify emergent properties. In a first step, we have taken the blinker and the traffic light of the game of life as examples to verify emergence by using refinement techniques; in a second step, since the refinement is not straightforward, we have taken another example, the Boids model, and by using timed automata and UPPAAL model checking techniques, we have been able to simulate and verify emergent properties.

Povzetek: Prispevek podaja pregled emergentnih lastnosti v kompleksnih sistemih.

1 Introduction

Technological development in various domains, especially the decrease in size and cost of electronic devices, like microprocessors and storage devices, leads to the development of distributed, decentralized, complex systems and systems of systems (SoS) [6]. A complex system is any system characterized by the interconnection of a large number of components with non-linear aggregate behaviour, *i.e.*, global behaviour is not derivable from the sum of the local or individual behaviours of components. Generally, these components have limited access or mostly they do not have any access to global information, which makes them operate on information obtained from the interaction with other components. Hence, their complexity increases with the number of components, in particular, when these later are highly interconnected.

This kind of what we call complex systems often exhibits properties that are not easily predictable by analysing the behaviour of their individual interacting components. In other words, complex systems exhibit emergent behaviours at the macro-level: behaviours that cannot be inferred from local behaviours or from behaviours at micro-levels. When engineering such systems, the verification of emergent properties is considered as a difficult issue for designers. These properties that cannot be deduced from components or systems when considered in isolation can be beneficial or harmful. Simulation is a good and powerful manner to detect emergence, but it is still not sufficient (a random exploration of behaviour is not guaranteed to exhibit the emergent ones). Hence, the use of formal techniques to detect and verify emergence becomes mandatory. In this paper, we present an overview of some definitions of emergence in the literature giving a distinction between weak and strong emergence and providing some works that have been carried out for the verification of emergent properties. On the one hand, we have taken examples of weak emergent properties from the game of life and verified them using standard techniques of refinement. On the other hand, since refinement is very difficult to apply for some examples which makes it undesirable for most designers, we have taken another example of the Boids model. By using timed automata [20, 21], we have been able to model, simulate and verify emergent properties using UPPAAL model checking techniques [19].

2 Emergence

The concept of emergence is widely used in various domains, and has different significations, from philosophical, social sciences, arts, biology to computer science [17, 25, 34, 35]. In the following, we will provide some definitions in the area of computer science, in particular in complex systems and SoS.

2.1 Definitions of emergent property

Despite several researches that have been done on the domain of complex system and SoS focusing on emergence, there is no consensus on the definition of emergence [1, 7, 25, 26]. However, many works are inspired from Aristotle's definition: *"the whole is greater than the sum of its parts".* An emergent property can be defined as "a property of an assemblage that could not be predicted by examining the components individually [10, 12]. Paul Teller [1, 8] said: "a property of a whole is an emergent property of a whole when it is not reducible to the non-relational properties of the parts ". In [1], Kopetz et al. preserve Paul's definition replacing the word "property" by "phenomenon": a phenomenon of a whole at the macro-level is emergent if and only if it is new with respect to the non-relational of any of its proper parts at the micro-level. In [2], Yong Meng defined emergent properties in complex systems as properties that are irreducible from knowledge of the interconnected components. In [5], Polack and Stepney define emergence as a discontinuity between global and local system descriptions. Sanders and Smith [6] defined emergent behaviour as a behaviour that is not determined by the behaviour of the constituents when considered in isolation. Isodora [4] characterized emergence of agents that operate in two or three-dimensional space as "a pattern appearing in the agents' configuration at some instance during the operation of the system".

2.2 Weak and strong emergence

As it is presented in many works in the literature and from different views in different domains, emergence is not an absolute concept, it can be classified on a scale from weak to strong [9].

2.2.1 Weak emergence

Weak emergence is a type of emergence in which the emergent property is amenable to computer simulation, it describes new properties arising in systems as a result of the interactions at an elemental level. However, it is stipulated that the properties can be determined by observing or simulating the system, and not by any process of a priori analysis [1, 10, 11]. *I.e.*, weak emergence is deducible but unexpected from the laws of the low-level domain. Bedau [11] describes deducible features of weak emergence in terms of derivability by simulation.

2.2.2 Strong emergence

Strong emergence is a type of emergence in which the emergent property cannot be simulated by a computer. It describes the direct causal action of a high-level system upon its components; qualities produced this way are irreducible to the system's constituent parts, *i.e.*, there is no theory, concept, or principle that can explain or deduce the behaviour of the system based on the properties or behaviour of its micro level components (the whole is greater than the sum of its parts). It follows that no simulation of the system can exist, for such a simulation would itself constitute a reduction of the system to its constituent parts [1, 10, 11].

3 Related work

In this section, we will show some research works in which emergence has been studied.

In [1], Kopetz et al. explained the phenomenon of emergence in systems of systems (SoS)[13]. Compared to monolithic systems, they noticed that emergent phenomena are the most differentiating characteristic of an SoS. They elaborated a short overview of emergence in two fields. In the field of philosophy: "how the emergence of a new properties of complex systems, such as life or the mind can be explained?". To answer this question, there were basically two camps. The first is the reductive physicalism view which claims that physical sciences can explain all that exists in the world. The second holds the opposite view of non-reductive physicalism, which claims that some new properties at the higher level cannot be reduced to mere physical phenomena.

In the domain of computer science, Kopetz et al. said that there is no consensus of emergence according to the remark of John Holland, a computer scientist working in the area of complex systems, saying *"Despite its ubiquity and importance, emergence is an enigmatic and recondite topic, more wondered at than analysed... It is unlikely that a topic as complicated as emergence will submit meekly to a concise definition and I have no such definition to offer"* [14].

In the European project TEREA SoS [1, 33], a roadmap for future research in SoS has been proposed, where the topics of theoretical foundation of SoS and emergence are in prominent position. They distinguished four cases in addition to the two types of emergence (weak and strong): expected and beneficial so emergent behaviour is a normal case; unexpected and beneficial: emergent behaviour is a positive surprise; expected and detrimental: emergent behaviour can be avoided by adhering to proper design rules; and unexpected and detrimental: emergent behaviour is a problematic case or a catastrophe.

They noticed that the type of emergence that is occurring in an SoS is weak emergence even if we are surprised and cannot explain the occurrence of an unexpected emergent phenomenon at the moment of its first encounter. After lighting both types of emergence phenomena, Kopetz et al. explained the causality of emergent phenomena in an SoS, finally, they proposed a new methodology for the design of complex systems based on the explanation of emergence engineering.

In [2], Yong Meng et al. proposed a grammar-based approach for verifying emergence in multi-agent systems. According to some works on emergence [7, 15, 16], they said that complex systems exhibit emergent properties that are irreducible from knowledge of the interconnected components. They provided an overview of emergence in three main perspectives: philosophy, natural and social sciences and computer science.

Computer science aims to predict, verify, validate and reason about emergence. Prediction is done before the observation of emergence, while verification is done only when emergence is observed. Verification of emergent properties is a complex task because it may lead to combinatorial explosion in terms of the system states. More importantly, emergent properties should be defined first before verifying. Validation allows determining whether an emergent property is beneficial or harmful [18]. Finally, reasoning enables the understanding of the cause-and-effect of emergence. Yong Meng et al. noticed that the only studied type of emergence in computer science is weak emergence. They classify formalization of emergence in three mainly categories: Variable-based, event-based and grammar based. They extended Kubik's grammar-based approach [17], which does not require a prior definition of emergence, in order to derive emergent property states.

In [3], Rouff et al. made an investigation into formal methods techniques that can be applicable to swarm based missions. They made a survey of some works in formal methods and extract a few ones for applying them in NASA swarm based systems. NASA future mission 2020-2030 will be the prospective ANTS (Autonomous Nano Technology Swarm). They argued that formal methods are very interesting to use for assuring the correctness of this mission. One of the most challenging aspects of using swarms is how to verify that the emergent behaviour of such a system will be proper and that no undesirable behaviour will occur. They cited several formal methods that have been used in the literature, but very few of them are used for verifying emergent behaviour of swarms. Based on the survey results, they selected four methods to do sample specification of parts of the ANTS mission and provided some advantages and inconveniences for each one. Finally, they suggested necessary properties of formal methods in order to specify swarm-based systems:

– modelling and reasoning about aggregate behaviour based on future actions of the individual agents of a swarm [27]

– modelling states of an agent of the swarm to assure correctness[28]

– modelling and reasoning about concurrent processes for detection of race conditions[29]

– modelling and reasoning about persistent information for verifying adaptive behaviour [28].

They conclude that the use of methods together is the best approach for the specification and the analysis of swarm-based systems.

In [4], Isodora et al. claimed that the verification of emergent behaviour of multi-agent system (MAS) is a very complex task, due to the fact that emergent behaviour must be detected and identified before starting its verification process. Unfortunately this identification is not an easy thing that requires the combining of formal and informal methods in order to verify MAS. They took an example of the aggressor defender game and proposed a research framework starting by formal modelling of agents which distinguish between spatial behaviour and other behaviours. Firstly, spatial behaviour leads to visual animation which can help to observe potential emergent properties. Then, spatial behaviour together with other behaviours can lead towards simulation and saving time series data. These can be used to identify patterns of behaviours combined with visual animation that produces desired properties. The desired properties, including emergence, can be verified in the original spatial agent model by model checking. They presented a superficial framework without details as well as a definition of spXMachine and a tool for automatic transformation to Net Lego (only results of animation were shown). They proposed future steps for this framework using some tools like spXM, FLAME [30] and DAIKON [31].

In [5], Polack and Stepney said that there is a discontinuity between global and local system descriptions for emergent systems which presents a challenge in terms of demonstrably-correct development (refinement) from an abstract specification. They gave a short review of refinement, which is a relationship between an abstract program and an equivalent concrete one. Such a refinement is discharged by simulation. The refinement proof must demonstrate that functional properties in the abstract model preserved in the concrete model. They took the example of cellular automata producing gliders. They argue that the behaviour rather than the movement of the glider cannot be refined to the rules of the game of life. They justified their argument by the disparate of languages in which specification and implementation are expressed. Finally, they gave tentative design guidelines for emergent systems. The first two guidelines are to identify the three key elements of the emergent system: required system specification, functional component specification and specification of the integration representation and to identify elements with common vocabulary and then identifying intermediate elements. They proposed a guideline to establish how the emergence is detected.

In [6], Sanders and Smith provided an approach for refining emergent properties. They said that during the modelling of an existing system, unknown discontinuities in behaviour may not be modelled. Consequently, proof techniques cannot be successful in detecting emergent behaviour. In addition, they noticed that when engineering new systems, we are not trying to prove the existence of emergent behaviour, we merely commence with the required emergent behaviour, which can be undesired behaviour that we must avoid, and then develop a design which gives rise to these new systems. The emergent behaviour must be a consequence of the component interactions within the design.

In [6], Sanders and Smith are only interested in systems with weak emergence. Their challenging effort was an answer for the question: Are standard formal methods and in particular refinement applicable to the engineering of systems with weak emergence? This question had been negatively answered in [5]. They took the same example of the glider to show how a positive answer for the above question can be made.

They started by specifying a simple game of life in one dimension and gave a specification of one-dimensional glider with its suitable implementation. Then they proved that such an implementation was a refinement of the specification. They demonstrated three examples (glider, floater and k-glider) in one-dimensional game of life that can be refined. The second step was a two-dimensional game of life. They took the same example of a glider specified in two dimensions using some helpful geometric results: They defined a rectangle in the plane as a Cartesian product of two finite intervals. A subset of the plane is said to be bounded iff it is contained in some rectangle, basing on this, they gave conditions for which a bounded sub set has a heading. According to this, they were able to prove that the two-dimensional implementation of the glider was the refinement of its specification, which is a positive challenging answer to what many researchers thought negatively. Hence, they argued that standard refinement is widely applicable for systems with weak emergent behaviours.

4 Refining emergent properties

In this section, in addition to Sanders' approach[6] to which we will add the time concept, we will take two different examples of weak emergent properties in the game of life in order to verify them using refinement techniques.

4.1 Game of life

The game of life [24], developed by the British mathematician J. H. Conway from the university of Cambridge in 1970, is a zero player game, its evolution is determined only by its initial configuration (initial state). Starting with the initial configuration we can see the game evolving during time.

4.1.1 Rules of the game

To update the state of the cell in the game, four rules are used:

1. Any live cell with fewer than two live neighbours die by isolation (solitude) or under-population.
2. Any live cell with more than three live neighbours die by overcrowding or over population
3. Any live cell with two or three live neighbours lives on to the next generation (survives).
4. Any dead cell with exactly three neighbours becomes a live cell as if by reproduction.

4.1.2 Game model

We model Conway's game of life using cellular automata in a two-dimensional infinite space. A cell has a state: a live cell is represented by true or 1, and a dead cell is represented by false or 0. For any cell, we use the Moore neighbourhood which is defined to consist of the current (central) cell and the eight cells surround it, as it is pictured in the figure 1.

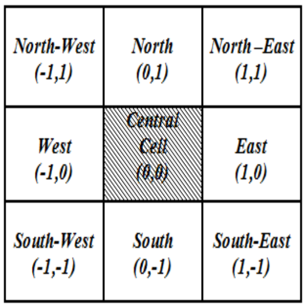

Figure 1: Moore neighbourhood cells with coordinates (x, y) on the plane

To describe shapes or patterns, consider a system whose state is a Boolean function on $\mathbb{Z}^2 \times \mathbb{N}$. At any discrete time $t : \mathbb{N}$, the state of the cell (n, m, t) in $\mathbb{Z}^2 \times \mathbb{N}$ is either true or false. We write

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

The state s can be updated according to the rules of the game, let $s\prime : \mathbb{B}$ be the updated state or the state after a transition of the state s.

Since states are updated synchronously in the game, we can write

$$s\prime = s[n, m, t + 1]$$

A cell has eight neighbours as defined previously in Figure 1. Let the central cell be the one whose coordinates in the two-dimensional infinite space are (l, c). (l for line and c for column). In order to explain behaviours in the game, we add time. Let $t : \mathbb{N}$ be the variable which represents discrete time. For a cell (l, c, t) we can write the Moore neighbours as follows:

$N(l, c, t) = \{(i, j, t) : \mathbb{Z}^2 \times \mathbb{N} \mid$
$(\mid i - l \mid < 2) \wedge (\mid c - j \mid < 2)\}$

that means a set of nine cells, the central one and the eight others surrounding it. In such a set, states are not taken into consideration.

In order to update a cell state at time t, the key operation is to calculate the number of live adjacent cells of this one at time t. Let $\Omega(l, c, t)$ be the function that calculates this number.

$$\Omega(l, c, t) : \mathbb{Z}^2 \times \mathbb{N} \rightarrow \mathbb{N}$$

$$\Omega(l, c, t) = \sum_{\substack{l-1 \leq i \leq l+1 \\ c-1 \leq j \leq c+1 \\ i \neq j}} s[i, j, t]$$

That can be specified in CSP as follows, [22, 23].

$$\Omega(l, c, t) := \#\{(i, j, t) \in N(l, c, t) \mid (i, j) \neq (l, c) \wedge s[l, c, t]\}$$

Let $\overline{\Omega}$ be the complement of Ω in 3×3 array which denotes the number of the dead adjacent cells of the central cell at time t.

$$\overline{\Omega}(l, c, t) = 8 - \Omega(l, c, t)$$

In order to update cells, first, we specify the four game rules in a formal manner.

Rule one: Any live cell with fewer than two live neighbours dies, that means:

$$\forall\, (l, c) \in \mathbb{Z}^2, t \in \mathbb{N} \cdot s[l, c, t] \wedge \Omega(l, c, t) < 2 \Rightarrow \neg\, s[l, c, t+1] \tag{1}$$

Rule two: Any live cell with more than three live neighbours dies

$$\forall\, (l, c) \in \mathbb{Z}^2, t \in \mathbb{N} \cdot s[l, c, t] \wedge \Omega(l, c, t) > 3 \Rightarrow \neg\, s[l, c, t+1] \tag{2}$$

Rule three: Any live cell with two or three live neighbours lives on to the next generation

$$\forall\, (l, c) \in \mathbb{Z}^2, t \in \mathbb{N} \cdot s[l, c, t] \wedge (\Omega(l, c, t) = 2 \vee \Omega(l, c, t) = 3) \Rightarrow s[l, c, t+1] \tag{3}$$

Rule four: Any dead cell with exactly three neighbours becomes a live cell

$$\forall\, (l, c) \in \mathbb{Z}^2, t \in \mathbb{N} \cdot \neg s[l, c, t] \wedge \Omega(l, c, t) = 3 \Rightarrow s[l, c, t+1] \tag{4}$$

To update cells, one of the four rules must be satisfied thus:

$(1) \vee (2) \vee (3) \vee (4)$ must be satisfied. From $(1) \vee (2)$ we write:

$$\forall\, (l, c) \in \mathbb{Z}^2, t \in \mathbb{N} \cdot s[l, c, t] \wedge (\Omega(l, c, t) < 2 \vee \Omega(l, c, t) > 3) \Rightarrow \neg\, s[l, c, t+1] \tag{5}$$

From $(3) \vee (4)$ we can write:

$$\forall\, (l, c) \in \mathbb{Z}^2, t \in \mathbb{N} \cdot \Omega(l, c, t) = 3 \Rightarrow s[l, c, t+1] \tag{6}$$

From $(5) \vee (6)$ we can write:

$$s[l, c, t+1] := s[l, c, t] \wedge \Omega(l, c, t) = 2 \vee \Omega(l, c, t) = 3 \tag{7}$$

The later formula (7) means that the cell at time t, on the line l and column c will be a live cell for the next generation at time $t+1$ for two cases:

- If it was a live cell at time t and has two live neighbours (it survives)
- It has three live neighbours(it survives if it was a live cell, or it gets born if it was a dead one).

Otherwise, this cell will be or will remain dead for the next generation.

4.1.3 Game state and initialization

The state of the game consists of the whole states of cells, which is an infinite set since the space is infinite.

Let I be the state which represents the initial configuration of the game and which assumed to be well-defined at the initial time t_0.

At the initial time, we assume that I is consisted of two sets; let I_0 be the one contains all live cells and $\overline{I_0}$ be the complement of I_0 in $\mathbb{Z}^2 \times \mathbb{N}$ that contains all dead cells, Thus, we write

$$I_0 := \{(i, j, t_0) \in \mathbb{Z}^2 \times \mathbb{N} \mid \ s[i, j, t_0]\}$$

$$\overline{I_0} := \{(i, j, t_0) \in \mathbb{Z}^2 \times \mathbb{N} \mid \ \neg s[i, j, t_0]\}$$

Assuming that I_0 is known, we can define the initial state of the game. The initialization is not easy to undertake because it depends on the purpose of the implementation and of the example to deal with; we have assumed that I_0 will be known in order to facilitate the treatment and the refinement of the following case studies.

4.2 Case study one: Blinker of period two

In this subsection, we will study a benchmark case known as a blinker of period two. Blinker is the smallest and most common oscillator, found by John Conway. The blinker pattern (behaviour) is like an oscillation of two line segments of period two. Only one line segment appears at time (line segments appear alternatively). To explain its behaviour, let us assume –as cited previously– that I_0 is known, which contains three consecutive live cells on the same line l at time t_0; and let the central cell be on column c

$$I_0 := \{(i, j, t_0) \in \mathbb{Z}^2 \times \mathbb{N} \mid \ s[i, j, t_0]\}$$

$$I_0 = \{(l, c-1, t_0), (l, c, t_0), (l, c+1, t_0)\}$$

$$\overline{I_0} := \{(i, j, t_0) \in \mathbb{Z}^2 \times \mathbb{N} \mid (i, j, t_0) \notin I_0\}$$

Using the transition rules, and by simulation, we can see the behaviour of the blinker at consecutive discrete times. Let I_1 be the set of all live cells after the first transition at time $t_1 = t_0 + 1$. According to the game rules (7) and to the simulation,

$$I_1 = \{(l-1, c, t_1), (l, c, t_1), (l+1, c, t_1)\}$$

Blinker behaviour is described as a switch between I_0 and I_1 during time $t : \mathbb{N}$; in addition, we can observe from simulation that I_0 consists of cells that constitute a horizontal line segment, whereas I_1 consists of cells that compose a vertical one; in other words, blinker behaviour will be described by the alternative appearance (emergence) of the horizontal line segment at even times and the vertical one at odd times.

Let I_n be the set of all live cells after $n : \mathbb{N}$ transitions at time $t_n = t_0 + n$. According to the game rules (7) and to the simulation, we can write

$$\forall\, t : \mathbb{N} \cdot\ t \mathit{\,mod\,} 2 = 0\ \Rightarrow I_t = I_0$$

$$\forall\, t : \mathbb{N} \cdot\ t \mathit{\,mod\,} 2 = 1\ \Rightarrow I_t = I_1$$

Figure 2 shows the blinker behaviour during discrete time.

If I_t, for $t : \mathbb{N}$, is a set of all live cells at time t, then ΘI_t will be the set of all live cells at the next generation (after one transition) at time $t + 1$.

From figure 2, on the one hand we can write

$$\Theta I_0 = I_1$$

$$\Theta I_1 = I_2$$

$$\Theta I_2 = I_3$$

$$\vdots$$

$$\Theta I_{n-1} = I_n$$

and on the other hand, we can write [1]

$$\forall\, t : \mathbb{N} \cdot\ I_t = I_0 \lhd\ even(t) \rhd\ I_1$$

$$\forall\, t : \mathbb{N} \cdot\ \Theta^t I_0 = I_0 \lhd even(t) \rhd I_1$$

$$\forall\, t : \mathbb{N} \cdot\ \Theta^t I_0 = I_{t \mathit{\ mod\ } 2}$$

$$\forall\, n, t : \mathbb{N} \cdot\ \Theta^n I_t = I_{(n+t) \mathit{\ mod\ } 2}$$

Informally, the behaviour of the blinker is represented by a horizontal line segment at even times and by a vertical one at odd times.

$$\forall\, (l, c, t), (l_1, c_1, t) \in \mathbb{Z}^2 \times \mathbb{N} \cdot$$
$$(l, c, t) \in I_0 \wedge (l_1, c_1, t) \in I_0 \Rightarrow (l = l_1)$$

$$\forall\, (l, c, t), (l_1, c_1, t) \in \mathbb{Z}^2 \times \mathbb{N} \cdot$$
$$(l, c, t) \in I_1 \wedge (l_1, c_1, t) \in I_1 \Rightarrow (c = c_1)$$

So far, we have explained the behaviour of the blinker of period two, now we specify it as follows

[1]Note that in CSP, ($a \lhd b \rhd c$) means if b then a else c.

$$\begin{gathered} Blinker := \exists\,(l, c) \in \mathbb{Z}^2,\ \forall\,(x, y) \in \mathbb{Z}^2, t \in \mathbb{N} \cdot \\ s[x, y, t] = |x - c| < 2 \wedge y = l \\ \lhd even(t) \rhd |y - l| < 2 \wedge x = c \end{gathered}$$

4.2.1 Initialization

To implement the blinker, we first initialize the game as expressed previously, so, at time $t = 0$, we put three consecutive live cells in the same line l, in such a way that the middle one assumed to be on the column c, [2],thus,

$$I_0 = \{(l, c-1, t), (l, c, t), (l, c+1, t)\}$$

represents all live cells at the initial time; all other cells are dead (infinite set represented by $\overline{I_0}$ as it is expressed above).

Since at the initial time, ($t = 0$) is even we simply write

$$init1 := \forall\,(x, y) \in \mathbb{Z}^2 \cdot$$

$$s[x, y, 0] = |x - c| < 2 \wedge y = l$$

4.2.2 Implementation

Previously we have seen that $s\prime$ is the state of s after one transition (update). According to the game rules, we can update cells as follows:

$$s[l, c, t+1] := s[l, c, t] \wedge \Omega(l, c, t) = 2 \vee \Omega(l, c, t) = 3$$

An easier choice of design expressed by a conditional transition knowing the positions of the central cell (line l and column c) is

$$s[x, y, t+1] = (y = l)\ \wedge |\ x - c|\ < 2$$

$$\lhd even(t)\ \rhd\ (x = c)\ \wedge\ |y - l|\ < 2$$

Thus, the implementation of the blinker of period two is as follows:

$$\begin{gathered} Bl1 := init1\ ;\ \textbf{do}\ true \rightarrow\ \forall\,(x, y) \in \mathbb{Z}^2, \\ t \in \mathbb{N}\ \cdot\ s\prime[x, y, t] := s[x, y, t+1]\ \textbf{od} \end{gathered}$$

4.2.3 Refinement

So far, we have specified and implemented the blinker of period two as it is expressed previously. Thus, $Blinker \sqsubseteq Bl1$ is a required refinement.

At the initial time $(t = 0)$ we have

$$\forall\,(x, y) \in \mathbb{Z}^2\ \cdot s[x, y, 0] = |x - c| < 2 \wedge y = l$$

which is assumed to be satisfied by the initialization for the condition at even time t, thus,

$$init1 = blinker[0/t].$$

[2]Note that it is possible that one can also initialize the blinker by putting three consecutive live cells on the same column c at time $t = 0$, in such a way that the middle one assumed to be on line l.

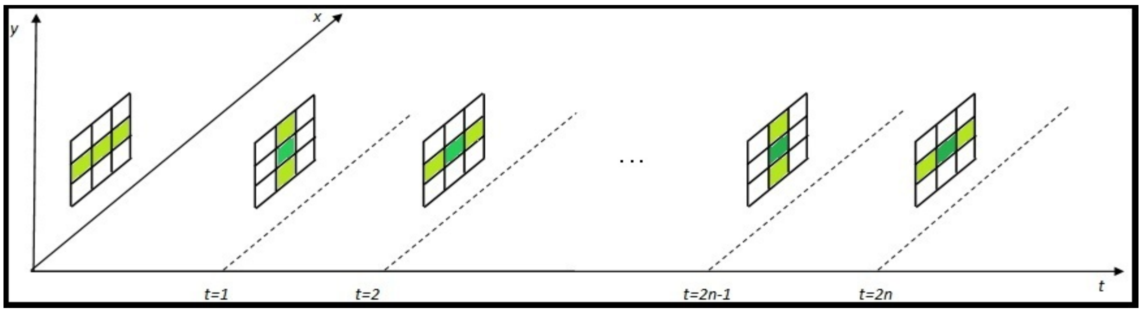

Figure 2: Blinker behaviour in the plane (x, y) during time t

Starting from the initial set at time $t_0 = 0$ we have

$$I_0 = \{(l, c-1, t_0), (l, c, t_0), (l, c+1, t_0)\}$$

At the initial state, all the live cells are located on the line l. According to the game rule specified previously by the formula(7) we have:
$\forall\, (l, c) \in \mathbb{Z}^2 \mid (t = 0) \wedge ((l > 1) \vee (l < -1)) \wedge$
$((c > 1) \vee (c < -1)) \cdot \Omega(l, c, t_0) < 2$
$\Rightarrow \forall\, (l, c), (x, y) \in \mathbb{Z}^2 \mid (t = 1) \wedge |y - l| > 1 \vee |x - c| > 1 \cdot \neg s[l, c, t_0 + 1]$

It means that all the cells located outside the (3×3) box whose central cell is $(l, c, t_0 + 1)$ are dead.
Now, if we look inside this box, at time t_0 we have only three live cells; according to the preceding definition of Ω and by using the rule(7) we can write
$\Omega(l, c, t_0) = 2 \wedge\ s[l, c, t_0]\ \Rightarrow s[l, c, t_0 + 1]$
$\Omega(l-1, c, t_0) = 3 \Rightarrow\ s[l-1, c, t_0 + 1]$
$\Omega(l+1, c, t_0) = 3 \Rightarrow\ s[l+1, c, t_0 + 1]$
$\Omega(l, c-1, t_0) = 1\ \Rightarrow\ \neg s[l, c-1, t_0 + 1]$
$\Omega(l, c+1, t_0) = 1 \Rightarrow\ \neg s[l, c+1, t_0 + 1]$
$\forall\, (l, c) \in \mathbb{Z}^2 \mid (t = 0) \wedge (|l| = |c| = 1) \Rightarrow\ \neg s[l, c, t_0 + 1]$

Let I_1 be the set of all live cells at time $t_1 = t_0 + 1$, thus

$I_1 = \{(l-1, c, t_1), (l, c, t_1), (l+1, c, t_1)\}$
$\Leftrightarrow \exists (l, c) \in \mathbb{Z}^2, \forall\, (x, y) \in \mathbb{Z}^2,\ \exists\, (t_1 = t_0 + 1) \in \mathbb{N} \cdot |y - l| < 2 \wedge\ x = c$

Now, starting from I_1 and by the same approach, we can write
$\forall\, (l, c), (x, y) \in \mathbb{Z}^2 \mid (t = 2) \wedge |y - l| > 1 \vee |x - c| > 1 \cdot \neg s[l, c, t_1 + 1]$
$\Omega(l-1, c, t_1) = 1\ \Rightarrow\ \neg s[l-1, c, t_1 + 1]$
$\Omega(l+1, c, t_1) = 1 \Rightarrow\ \neg s[l+1, c, t_1 + 1]$
$\forall\, (l, c) \in \mathbb{Z}^2 \mid (t = 1) \wedge |l| = |c| = 1 \Rightarrow\ \neg s[l, c, t_1 + 1]$
$\Omega(l, c, t_1) = 2 \wedge\ s[l, c, t_1]\ \Rightarrow s[l, c, t_1 + 1]$
$\Omega(l, c-1, t_1) = 3 \Rightarrow\ s[l, c-1, t_1 + 1]$
$\Omega(l, c+1, t_1) = 3 \Rightarrow\ s[l, c+1, t_1 + 1]$

Let I_2 be the set of all live cells at time $t_2 = t_1 + 1$, thus
$I_2 = \{(l, c-1, t_2), (l, c, t_2), (l, c+1, t_2)\}$
$\Leftrightarrow \exists\, (l, c) \in \mathbb{Z}^2, \forall\, (x, y) \in \mathbb{Z}^2,\ \exists\, (t_2 = t_1 + 1) \in \mathbb{N} \cdot |x - c| < 2 \wedge\ y = l$

If we look to the sets in which the time is even (I_0 and I_2), we can observe that both contain the same live cells, consequently, it will be the same for odd times for I_1 and I_3. Generalizing time over $\mathbb{N}$, we simply infer the blinker specification.

Now, we assume that the blinker condition is satisfied at time t, and we prove that it will hold at time $t + 1$, thus, we assume that

$$s[x, y, t + 1] = |x - c| < 2 \wedge y = l \\ \lhd\ even(t) \rhd |y - l| < 2 \wedge x = c$$

is satisfied. At time $t + 1$ we have

$$s[x, y, (t + 1) + 1] = |x - c| < 2 \wedge y = l \\ \lhd\ even(t + 1) \rhd |y - l| < 2 \wedge x = c$$

Obviously, if t is even then $t{+}1$ is odd and hence the second conditional expression holds, and when t is odd, the first expression of the condition holds as well. Replacing t by $t + 1$ we generalize over $\mathbb{N}$ to infer the blinker period-two specification.

4.2.4 Another implementation

Another choice of implementation is that one may think that the behaviour of blinker during time is like a composition of two functions (see Figure 2). The first one takes the set of live cells and translates each cell (location) in the plane by one unit of time, the second takes the result of the first and rotates each cell (except the central one) by an angle θ equals ninety degrees in counter-clockwise direction (this direction is chosen for simplicity) [3].
Preserving the cell state and taking the same initialization expressed previously in section (4.2.1), the blinker can be implemented as follows

$$Bl2 := init1\ ;\ \textbf{do}\ true \rightarrow\ \forall\, x, y : \mathbb{Z}, \\ t : \mathbb{N} \cdot\ \ s[-y, x, t + 1] := s[x, y, t]\textbf{od}$$

4.2.5 Refinement

The required refinement is $blinker \sqsubseteq\ Bl2$

[3]Note that the clockwise direction is possible for design, in addition, alternative directions are also possible

At the initial time, it is the same as (4.2.3)

$$init1 = blinker[0/t].$$

Using the same approach as the preceding refinement, based on the formula(7), we can obtain sets I_0, I_1 which consist of cells representing a perpendicular line segments sharing the same cell centre in successive times.
At time $t+1$, we have

$$s\prime[x,y,t] = s[x,y,t+1] = s[-y,x,t+1]$$

In order to make an easier proof, we assume that $l = c = 0$. Assume that τ be the function translating the cell location by one time unit, and let ρ be the function that rotates the cell by 90 degrees; thus

$$\rho := \mathbb{Z}^2 \times \mathbb{N} \to \mathbb{Z}^2 \times \mathbb{N}$$
$$\rho\,(x,y,t) = \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ t \end{pmatrix}$$
$$\tau :\ \mathbb{Z}^2 \times \mathbb{N} \to \mathbb{Z}^2 \times \mathbb{N}$$
$$\tau(x,y,t) = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} x \\ y \\ t \end{pmatrix} = \begin{pmatrix} x \\ y \\ t+1 \end{pmatrix}$$

The composition of these functions is commutative.

$$\rho \circ \tau : \mathbb{Z}^2 \times \mathbb{N} \to \mathbb{Z}^2 \times \mathbb{N}$$
$$\rho \circ \tau(x,y,t) = \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ t+1 \end{pmatrix}$$

Since the angle θ equals $90°$, then we have

$$\rho \circ \tau(x,y,t) = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ t+1 \end{pmatrix}$$
$$= \begin{pmatrix} -y \\ x \\ t+1 \end{pmatrix}$$

$$s\prime[x,y,t] = s\,[-y,x,t+1] = s[x,y,t+1]$$

We can see that at successive times, line segments are always perpendicular, which satisfies the blinker specification ($y = l$) when t is even and ($x = c$) for the other case; whereas the bounded conditions ($|y-l| < 2$) and ($|x-c| < 2$) are satisfied at the moment of the initialization.

4.3 Case study two: traffic light

The traffic light is a well known example in the game of life. It is an oscillator formed by four synchronous blinkers of period two that do not encounter each other. The four blinkers of different directions (East, West, North and South) are bounded by a box of 9×9 cells. More precisely, they surround a box of 3×3 dead cells. If we consider $d = (l,c)$, (l for line and c for column) to be the central cell of these boxes, then the central cells of both blinkers (East and West) are on the same line l and symmetrical with respect to the cell d, the other blinkers (North and South) have central cells on the same column c and symmetrical with respect to d, as illustrated in Figure 3.

4.3.1 Specification

First, we need to specify the four blinkers, each in isolation from the others. Let $(l_e, c_e), (l_w, c_w), (l_n, c_n), (l_s, c_s)$ be the central cells for the east, west, north and south blinkers respectively.

Since the central cells of the east blinker $Eblinker$, and the west blinker $Wblinker$, are located on line $l_e = l_w = l$ and column $c_e = c+3$ and $c_w = c-3$ respectively, we have

$Eblinker := \exists\, l,c : \mathbb{Z},\ \forall\, x,y : \mathbb{Z}, t : \mathbb{N} \cdot$
$s[x,y,t] = (|x-c_e| < 2) \wedge (y = l)$
$\lhd even(t) \rhd (|y-l| < 2) \wedge (x = c_e)$

$Wblinker := \exists\, l,c : \mathbb{Z},\ \forall\, x,y : \mathbb{Z}, t : \mathbb{N} \cdot$
$s[x,y,t] = (|x-c_w)| < 2) \wedge (y = l)$
$\lhd even(t) \rhd (|y-l| < 2) \wedge (x = c_w)$

Since the central cells of the north blinker $Nblinker$ and the south blinker $Sblinker$ are located on the column $c_n = c_s = c$ and the line $l_n = l+3$ and $l_s = l-3$ respectively, we have

$Nblinker := \exists\, l,c : \mathbb{Z},\ \forall\, x,y : \mathbb{Z}, t : \mathbb{N} \cdot$
$s[x,y,t] = (|y-l_n| < 2) \wedge (x = c)$
$\lhd even(t) \rhd (|x-c| < 2) \wedge (y = l_n)$

$Sblinker := \exists\, l,c : \mathbb{Z},\ \forall\, x,y : \mathbb{Z}, t : \mathbb{N} \cdot$
$s[x,y,t] = (|y-l_s| < 2) \wedge (x = c)$
$\lhd even(t) \rhd (|x-c| < 2) \wedge (y = l_s)$

So far, each blinker is specified in isolation. A traffic light $Tlight$ is a synchronous composition of the four blinkers specified previously.

$Tlight := Eblinker \wedge Wblinker \wedge Nblinker \wedge Sblinker$

$Tlight := \exists\, l,c : \mathbb{Z},\ \forall\, x,y : \mathbb{Z}, t : \mathbb{N} \cdot$

$(|x-c_e| < 2) \wedge (y = l) \bigwedge (|x-c_w| < 2) \bigwedge (|y-l_n| < 2) \wedge (x = c) \bigwedge (|y-l_s| < 2)$

$$\lhd\ even(t) \rhd$$

$(|y-l| < 2) \wedge (x = c_e) \bigwedge (x = c_w) \bigwedge (|x-c| < 2) \wedge (y = l_n) \bigwedge (y = l_s)$

Replacing c_e by $(c+3)$, c_w by $(c-3)$, l_n by $(l+3)$, and l_s by $(l-3)$ we obtain

$Tlight := \exists\, l,c : \mathbb{Z},\ \forall\, x,y : \mathbb{Z}, t : \mathbb{N} \cdot$

$(|x-c-3| < 2) \wedge (y = l) \bigwedge (|x-c+3| < 2) \bigwedge (|y-l-3| < 2) \wedge (x = c) \bigwedge (|y-l+3| < 2)$

$$\lhd\ even(t) \rhd$$

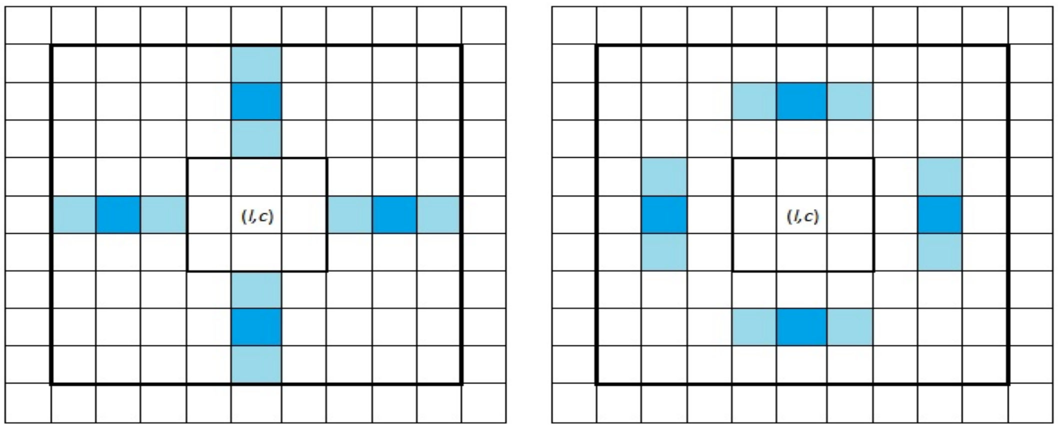

Figure 3: Traffic light behaviour: from the left to the right : behaviour at even and odd times respectively

$(|y - l| < 2) \wedge (x = c + 3) \bigwedge (x = c - 3) \bigwedge (|x - c| < 2) \wedge (y = l + 3) \bigwedge (y = l - 3)$

4.3.2 Initialization, implementation and refinement

The initialization, the implementation as well as the required refinement could be inferred easily from the preceding Blinker period-two example.

5 Verification of emergent property using model checking

A well known example of weak emergence is the Boids model, which captures the motion of flocking birds. Boids is an artificial life program, developed by Craig Reynolds in 1986, which simulates the flocking behaviour of birds [32]. In our case study, we propose a number of ducks that can swim together in a lake. Each duck can be anywhere in the lake moving in different directions with various speeds. The global behaviour (at the macro level) is a movement (swimming) of ducks together forming a flock. At the micro level (local behaviour), each duck swims according to three rules:

1. Alignment: swim towards the average heading of local flock-mates
2. Separation: swim to avoid crowding neighbours
3. Cohesion: swim to move towards the average position (centre of mass) of local flock-mates.

5.1 System model

To demonstrate our method, we have taken the Boids model as in [2]. We have taken a set of similar ducks swimming in a large lake (represented by an infinite two-dimensional space of cells). In order to capture the behaviour of the set of ducks, we modelled the system as a set of processes behaving in concurrency. Each process is modelled by a timed automaton. The calculation of the distance between ducks as well as speeds and directions is done by another process, called a *"controller"*, which gets information from all ducks then informs them about its state (*Close*, *Far* or *Collision*). The *controller* is used just for some calculations needed by ducks.

Initially all ducks have random positions on the lake. A duck may take from an initial state and according to the distance between other ducks, another state that can be in the state *Collision* when it runs into at least another duck (distance is nil or smaller than a minimal value); it may approach the flock and go to the state *Close* when it is in the flock (distance is between a minimal and maximal value); and it can go to a *Far* state when it is far from the flock (distance is greater than a maximal value).

In order to make an easier model, we have assumed that a duck can move with only two speeds, (one or two cells per time unit). It can slow down when it approaches the flock in order to keep the average speed of the flock. It may speed up to reach the flock when it is far from it. In addition to the eight directions implemented in [2], we have modelled all possible directions to move from a current cell to a neighbour one, hence, sixteen directions are allowed.

A cell c_{ij} is considered a neighbour to a cell c_{mn} if and only if
$|i-m| < 3 \wedge |j-n| < 3$. Hence, a cell has 24 neighbours.

Figure 4 shows the general duck process represented by timed automata in UPPAAL.

At the *Initial* state, a $duck_i$ has a position in the lake, waiting for an event from the $controler$ in one of the three channels (far, close, collision). Depending on which channel the duck receives an event, it changes the state updating the speed (speed up, slow down). The state *"Collision"* is disliked, it will make the duck in hazard, consequently it deadlocks the system. In the other two states, the duck is always waiting for an event from the controller in the three channels cited above.

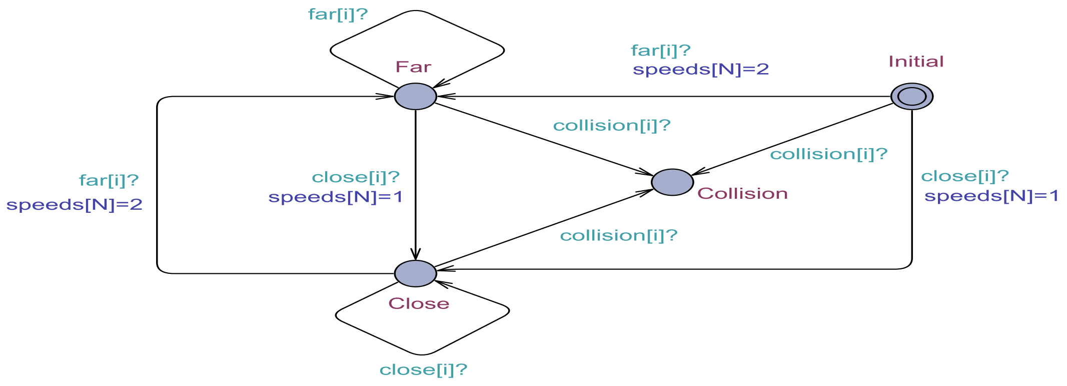

Figure 4: Timed automata modelling the general process of duck

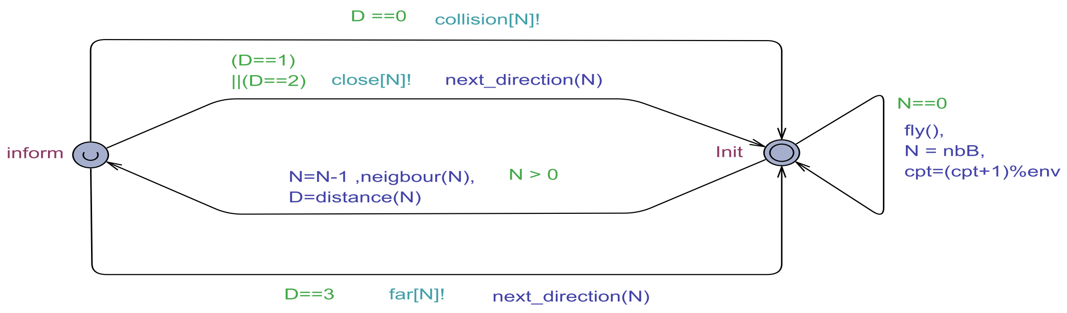

Figure 5: Timed automata modelling the controller process

The process *controller* (see Figure 5), has two states. At the initial state, it chooses a duck and determines its neighbours and calculates the distance separating it from the other ducks. After that, it goes to an urgent location[19]. Depending on the value of that distance, it informs the duck in question about its state via channels. These operations are repeated for all ducks.

After finishing the calculation for the whole set of ducks, the *controller* informs them to fly in a synchronous manner, updating their positions and directions and consequently updating their velocities. The process *controller* will be executed infinitely.

A duck's position is calculated taking in account the duck's speed and direction, whereas the direction (alignment) is deduced from the sum of all direction vectors of neighbours of the duck if it is in a *Close* state. When the duck is in a *Far* state, it will take the direction of the vector that starts from the current position toward the centre of the flock. Since the duck has an internal separation rule, it can change the direction as well as the speed one or many times, in order to forbid collision with other ducks whenever there is a free neighbour position.

5.2 Simulation and verification

We have implemented the system model based on timed automata using the UPPAAL model checker. Running the processes in concurrency and starting simulation with UPPAAL, we see that ducks starting from different initial positions (that can be a known configuration of ducks by the designer at initial time) can be all at the *Close* state. This represents a property of a whole (global property at the macro level), meaning that ducks will swim together, which represents an emergent property that we would like to verify.

Using the UPPAAL verifier, we have checked for some safety and emergent properties. For the safety property, we have checked the system deadlock $A[]not\ deadlock$, and executed the query that all ducks never reach a collision state. $A[]forall(i : id\_B)\ !Bird\_body(i).Collision$.

For the emergent property, we checked that all ducks swim one close to each other or ducks form a flock. In this paper we are only interested in this property; we have executed the query that eventually all ducks can reach simultaneously the state *Close*, $E <> forall(i : id\_B)\ Bird\_body(i).Close$. We changed the number of ducks in the model using a laptop of dual processors with 2.2 G. Hertz each and 4 G. Bytes of RAM. The results are summarized in Table 1.

6 Conclusion

The use of formal methods, in particular formal verification, becomes very important for engineering complex systems and SoS. Simulation is very useful for detecting emer-

Number of ducks	Property	Verification	Kernel	Elapsed time used
2	Deadlock	0.14	0.031	0.234
	Collision	0.062	0.031	0.134
	Emergence	0.0	0.0	0.016
3	Deadlock	0.25	0.031	0.312
	Collision	0.187	0.0	0.234
	Emergence	0.0	0.0	0.015
4	Deadlock	0.749	0.062	0.826
	Collision	0.374	0.031	0.405
	Emergence	0.0	0.0	0.016
5	Deadlock	2.137	0.016	2.153
	Collision	1.108	0.078	1.263
	Emergence	0.0	0.0	0.0
10	Deadlock	4.118	0.093	4.29
	Collision	1.748	0.078	1.841
	Emergence	0.0	0.0	0.0
20	deadlock	112.414	0.297	113.568
	Collision	48.329	0.343	48.894
	emergence	0.0	0.0	0.0
40	Deadlock	explosion	/	/
	Collision	explosion	/	/
	Emergence	0.016	0.0	0.015
50	Deadlock	explosion	/	/
	Collision	explosion	/	/
	Emergence	0.016	0.0	0.02
60	Deadlock	explosion	/	/
	Collision	explosion	/	/
	Emergence	0.031	0.0	0.022
70	Deadlock	explosion	/	/
	Collision	explosion	/	/
	Emergence	0.032	0.0	0.024
90	Deadlock	explosion	/	/
	Collision	explosion	/	/
	Emergence	0.032	0.0	0.033
100	Deadlock	explosion	/	/
	Collision	explosion	/	/
	Emergence	0.047	0.0	0.056

Table 1: Verification time (in seconds) for properties per number of ducks: 0 means smaller than a millisecond or neglected time.

gent behaviours but is not sufficient for ensuring system correctness, especially for critical systems, when emergent properties are detrimental. Unfortunately, verifying such systems is not straightforward, not only because of the large number of components or sub-systems that are highly interconnected which leads to state explosion during the model checking, but also due to unknown and unexpected behaviours that can be beneficial or harmful leading to a fault in the system. The technique of refinement based on mathematical theory is very powerful, but, on the one hand, it is still undesirable for designers because of their difficulties of understanding, and on the other hand, it is not obvious for complex examples. The use of model checking techniques which needs some experiences and knowledge of the model in question ensures the correctness of the system design. Unfortunately, model checking suffers from state explosion when the system is composed of a big number of constituents or sub-systems. The method developed in this paper based on the Uppaal model checker shows very good results for verifying emergent properties. Unfortunately it has limitations for the verification of deadlock properties when the number of sub-systems becomes important.

Acknowledgement

The authors would like to thank the ministry of higher education and scientific research of Algeria, as well as the institute of research in computer science IRIT, Paul Sabatier University, Toulouse, France, for supporting this research work.

Our sincere thanks also goes to Dr. Rudolf Schlatte from the university of Oslo, for his insightful remarks after reading this paper.

References

[1] H. Kopetz, O. Höftberger, B. Frömel, F. Brancati, F. Brancati (2015), Towards an understanding of emergence in systems of systems, *10th System of Systems Engineering Conference (SoSE)*, IEEE. `https://doi.org/10.1109/SYSTEMS.2008.4518983`

[2] Y. M. Teo, B. L. Luong and C. Szabo (2013), Formalization of Emergence in Multi-agent Systems, *Conference on Principles of Advanced Discrete Simulation (PADS)*, ACM SIGSIM, Montreal, Canada. `https://doi.org/10.1145/2486092.2486122`

[3] C. Rouff, A. KCSVanderbilt, W. Truszkowski, J. Rash, Mike Hinchey (2004), Verification of NASA emergent system, *International Conference on Robotics and Automation*, IEEE, 154, pp.231-238. `https://doi.org/10.1109/ICECCS.2004.1310922`

[4] I. Petreska, P. Kefalas, M. Gheorghie (2011), A framework towards the verification of emergent properties in spatial multi-agent systems, *Proceeding of the workshop on application of software agents*, ISBN 978-86-7031-188-6, pp. 37 -44.

[5] F. Polack and S. Stepney (2005), Emergent properties do not refine, *electronic notes in theoritical computer science* 137, pp. 163- 181. `https://doi.org/10.1016/j.entcs.2005.04.030`

[6] J.W. Sanders and G. Smith (2007), Refining emergent properties, *Electronic notes in theoretical computer science*, 259, pp. 207-223. `https://doi.org/10.1016/j.entcs.2009.12.026`

[7] J. Deguet, Y. Demazeau, L. Magnin (2006), Elements about the Emergence Issue: A Survey of Emergence

Definitions, *ComPlexUs*, 3, pp. 24-31.
https://doi.org/10.1159/000094185

[8] P. A. Teller (1992), Contemporary Look at Emergence. in: Beckermann, A. et al. (editors) Essays on the Prospects of Nonreductive Physicalism, De Gruyter .
https://doi.org/10.1515/9783110870084.139

[9] J. J. Johnson IV, A. Tolk, A. Sousa-Poza (2013), A Theory of Emergence and Entropy in Systems of Systems, *Procedia Computer Science 20* , pp. 283 – 289.
https://doi.org/10.1016/j.procs.2013.09.274

[10] M. A. Bedau (1997), Weak Emergence, Philosophical Perspectives: Mind, Causation, and World, 11, pp.375–399.
https://doi.org/10.1111/0029-4624.31.s11.17

[11] M. A. Bedau (2003), Downward causation and autonomy in weak emergence. Principia, 6, pp. 5–50.
https://doi.org/10.5007/%25x

[12] C.Szabo, Y. Meng Teo, G. K. Chengleput (2014), understanding complex systems: using interaction as a mesure of emegence, *Proceedings of the 2014 Winter Simulation Conference*, IEEE.
https://doi.org/10.1109/WSC.2014.7019889

[13] M. Jamshidi (2009), Systems of Systems Engineering, John Wiley and Sons.
https://doi.org/10.1002/9780470403501

[14] J. Holland (1998), Emergence, from Chaos to Order, Oxford University Press.

[15] V. Darley (1994), Emergent Phenomena and Complexity, *Artificial Life IV*, pp. 411-416.

[16] Z. Li, C. H. Sim, and M. Y. H. Low (2006), A Survey of Emergent Behaviour and Its Impacts in Agent-based Systems, *Proc of IEEE International Conference on Industrial Informatics*, pp. 1295-1300.

[17] A. Kubik (2003), Toward a Formalization of Emergence, *Artificial Life IX*, 9(1), pp. 41-65.
https://doi.org/10.1162/106454603321489518

[18] C. Szabo and Y. M. Teo (2012), An Integrated Approach for the Validation of Emergence in Component-based Simulation Models, *Proc of Winter Simulation Conference*, pp.1-12.

[19] G. Behrmann, A. David, K. G. Larsen (2004), A tutorial on Uppaal, Available at http://www.uppaal.com.
https://doi.org/10.1007//978-3-540-30080-9_7

[20] R. Alur (1999), Timed automata, *Proc. CAV'99*, Springer LNCS, 1633 pp.8-22.

[21] R.Alur and D. Dill (1994), A theory of timed automata, *Theoretical Computer Science*, 126 pp.183-235.
https://doi.org/10.16/0304-3975(94)90010-8

[22] A. W. Roscoe (1998), The Theory and Practice of Concurrency, Prentice-Hall.

[23] C.A.R. Hoare (1978), Communicating Sequential Processes, *Communications of the ACM*, ACM, 21(8)pp. 666-677.
https://doi.org/10.1145/359576.359585

[24] E. R. Berlekamp, J. H. Conway, R. K. Guy (1982), Winning Ways for your Mathematical Plays, *volume 2, Games in Particular*. Academic Press, London.

[25] E. Bonabeau, J-L. Desslles, A. Grumbach (1995), Characterizing emergent phenomen(1) : A conceptual framework, *Revue Internationale de Systemique, Vol. 9, N. 3.*

[26] E. Bonabeau, J-L. Desslles, A. Grumbach (1995), Characterizing emergent phenomena(2) : A critical review, *Revue Internationale de Systemique, Vol. 9, N. 3.*

[27] C. Tofts (1991), Describing social insect behaviour using process algebra, *Transactions on Social Computing Simulation*, pp. 227-283.

[28] J. Bamard, J. Whitworth, M. Woodward (1996), Communicating X-machines, *Journal of Information and Software Technology*.
https://doi.org/10.1016/0950-5849()9501066-1

[29] K. Chandy, J. Mism (1988), Parallel Program Design: A Foundation, Addison-Wesley.

[30] M. Pogson, R. Smallwood, E. Qwarnstrom, M. Holcombe (2006), Formal agent-based modelling of intracellular chemical interactions, Biosystems 85 pp. 37-45.
https://doi.org/10.1016/j.biosystems.2006.02.004

[31] D. E. Michael, G. G. William, K. Yoshio, D. Notkin (2000), Dynamically discovering pointerbased program invariants, *Technical Report UW-CSE-99-11-02, University of Washington Department of Computer Science and Engineering*, Seattle, WA

[32] Craig W. Reynolds (1987), Flocks, herds and schools: A distributed behavioural model, *SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, Association for Computing Machinery. pp. 25–34. https://doi.org/10.1145/37401/37406

[33] M. Henshaw et al. (2013), The Systems of Sysems Engineering Strategic Research Angenda, *TAREA-PU-WP5-R-LU-26*, Loughborough University United Kingdom.

[34] Serugendo G.D.M , M.P. Gleizes, A. Karageorgos (2006), Self-Organisation and Emergence in MAS: An Overview, *Informatica* 30(1),45-54.

[35] Serugendo G.D.M , M.P. Gleizes, A. Karageorgos (2005), Self-Organisation in multi-agent systems, *The knowledge Engineering Review* vol.20:2, 165–189, Cambridge University Press. https://doi.org/10.1017/S0269888905000494