Noname manuscript No.
(will be inserted by the editor)

Modelling Flocks of Birds and Colonies of Ants from the Bottom Up

Rocco De Nicola · Luca Di Stefano · Omar Inverso · Serenella Valiani

the date of receipt and acceptance should be inserted later

Abstract This paper advocates the use of compositional specifications based on formal languages as a means of modelling and analysing sophisticated collective behaviour in natural systems. With the use of appropriate linguistic constructs, models can be developed that are both compact and intuitive, and can be easily refined and extended in small steps. Automated workflows can be implemented on top of this methodology to provide quick feedback, enabling rapid design iterations. To support our argument, we present three examples from the natural world, focusing on flocks of birds and colonies of ants, which feature well-known examples of emergent behaviour in collective adaptive systems. We use an agent-based language to develop simple models that aim at capturing these collective phenomena, and discuss the specific language constructs that we use in the process. Then, we adapt an existing verification tool for the language to simulate our models, and show that our simulations do display emergent behaviour.

Acknowledgements Work partially funded by MIUR project PRIN 2017FTXR7S *IT MATTERS* (Methods and Tools for Trustworthy Smart Systems), ERC consolidator grant no. 772459 *D-SynMA* (Distributed Synthesis: from Single to Multiple Agents), and PRO3 MUR project *Software Quality*.

Rocco De Nicola
IMT School for Advanced Studies, Lucca, Italy

Luca Di Stefano
University of Gothenburg, Gothenburg, Sweden

Omar Inverso
Gran Sasso Science Institute (GSSI), L'Aquila, Italy

🖂 Serenella Valiani
IMT School for Advanced Studies, Lucca, Italy
E-mail: serenella.valiani@imtlucca.it

1 Introduction

Over the years, biological and natural systems such as flocks of birds, colonies of ants, schools of fish, and swarms of insects have received considerable attention from researchers across different disciplines. These systems exhibit complex structures that can dynamically respond to changing conditions. To understand these systems, researchers have used various mathematical frameworks. For instance, *flocking*, where a group of birds exhibits coherent patterns of collective motion, has been modelled using graph theory [36], distributed control laws [46], and statistical mechanics [6]. Another example can be found in colonies of ants; the way they distribute their workforce has been studied through differential equation [48], search algorithms and strategies [35, 53] or probabilistic communication [14]

The design of these systems has been approached in terms of aggregate features and relies on simplifying assumptions about the individual behaviour. However, recent studies have shown that a bottom-up design approach could lead to a more effective design of adaptive systems. Compositional methodology has gained prominence in several disciplines, including epidemiology, ecology, economics, and social sciences [30, 24, 47, 8], where the focus lies on the examination of individual components rather than the entire system.

In this paper, we argue for a bottom-up approach based on formal specification languages. This approach defines the system in terms of individual components and local rules, allowing the collective behaviour of the system to emerge naturally from the combined effect of the actions of the components. This approach can be helpful in reproducing sophisticated collective dynamics intuitively, and, when combined with appropriate linguistic constructs, can yield compact and intuitive specifications that are easy to refine. The adoption of a formal language allows the implementation of automated workflows for simulation or formal analysis that can provide quick feedback, enabling rapid design iterations.

We present three examples to illustrate our point and write our models using a specification language previously defined by us [11]. In the first example we develop a model of a flock by gradually defining the individual behaviour and features of a bird. As we progressively refine it, we aim at keeping the behaviour of individual birds as decentralized as possible. We gradually present the constructs used in the specifications, to keep them compact and intuitive. Upon attaining a fully refined model, we simulate the evolution of a flock obtained by composing a number of birds together. Our results indicate that the flock exhibits collective characteristics. In particular, when faced with external aggression from a predatory bird, the members of the flock demonstrate the ability to flee the threat and subsequently reform into a cohesive unit once the danger has passed. The second example introduces a model that describes the behaviour of an ant within a colony. We simulate the evolution of the behaviour of a colony during food foraging. In particular, we show that the colony tends to choose the shortest path when faced with multiple paths. The third example, on the other hand, describes the behaviour of a small group of ants on a finite bar. It is trivial to observe that the ants fall off the bar. The emergent behaviour we aim to demonstrate concerns the order of events that characterize the model. We show the order of events using formal verification techniques. We borrowed this last example from a paper presented at the conference to which this special issue is dedicated [23].

In our view, the considered examples provide evidence that the design of adaptive systems can benefit from the exploitation of the features and principles of biological and natural systems. By using a bottom-up approach, where the behaviour of individual entities is governed by simple rules and local reactions, it becomes possible to design a scalable, adaptive, and efficient framework to address the research questions posed in the field of adaptive systems design.

This paper is a revised version of [12] but extends it in several ways. In our previous paper, we considered the evolution of a single simulation of the model presented in Section 2.1, while here in Section 3.1, we provide further simulation results that show the level of cohesion that the flock is able to achieve after an attack. Furthermore, we introduce new examples (Sections 2.2 and 2.3) along with their experimental evaluation. In particular, in Section 3.2 we provide simulation results that show how a colony of ants distributes along two possible paths during the food foraging process. In Section 3.3 we show instead how we can prove interesting temporal properties (in this case, about the ordering of events for every execution of the example from Section 2.3) by means of mechanized verification procedures.

Listing 1: Baseline agent modelling.

```
agent Bird {
    interface =
        x: 0..G;
        y: 0..G;
        dirx: −D..D + 1;
        diry: −D..D + 1

    Behaviour = Move; Behaviour
    Move = {
        x ← x + dirx;
        y ← y + diry
    }
}
```

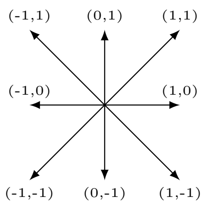
The paper is structured as follows. In Section 2, we define the three models of the flock of birds and ant colonies using various constructs of the language. Our experimental setup for simulation and verification, along with our controlled experiments, are described in Section 3. In Section 4, we provide an overview of related work. Finally, in Section 5, we conclude with some final remarks and discuss potential avenues for future research.

2 Specification

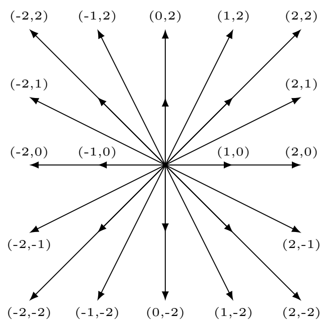
In this section, we present three examples that illustrate how to use LAbS to model agents. Starting with simple initial models, we gradually refine the specifications by leveraging the expressiveness of the language and its constructs, with the goal of obtaining intuitive and readable specifications of the individual agents in the system.

2.1 Flock of birds

The model in this Section mimics the dynamics of a flock of birds when confronted by a predator. Our model is constructed in stages, with the introduction of language constructs occurring as the model is expanded, thereby preserving a concise and intuitive specification.
Description of a bird. Each bird of the flock possesses two defining attributes; specifically, its *position* and its *orientation*. The former is represented by a set of coordinates in a two-dimensional space denoted as



(a) $D = 1$.



(b) $D = 2$ (some labels omitted for readability).

Fig. 1: Possible heading vectors that a bird can assume for different values of D.

(x, y), while the latter is characterized by a pair of integers $(\mathsf{dirx}, \mathsf{diry})$ that represent a heading vector. This comprehensive depiction enables the representation of both the bird's displacement direction, as indicated by the angle subtended by the heading vector, and its velocity, which is portrayed by the length of the heading vector.

Listing 1 shows how we can model the above description.[1] We start by defining and initializing the observable features, or *attributes*, of the agent. These are listed within the first Section named **interface**, in lines 2–6. Attributes x and y correspond to any valid coordinate on a grid that represents an arena where the flock is located. The grid is a square with edges of length G, so the possible values range from 0 to $G-1$ inclusive (lines 3–4). Currently, we assume that agents never reach the edge. The range of initial values for dirx and diry spans from $-D$ to D (lines 5–6). Here, D represents the maximum displacement along each coordinate of the grid. It is noteworthy that, as the value of D increases, the number of possible heading vectors also increases, as is illustrated in Figure 1. Finally, the actual initial value of each attribute is chosen non-deterministically.[2]

Behaviour of a bird. Listing 1 also specifies a very simple behaviour for our birds. The behaviour is defined as a Behavior process. Its definition is recursive, meaning that each bird will repeatedly carry out the actions described in the **Move** process (line 8). This process, in turn, updates the two attributes (x, y) with $(\mathsf{x}+\mathsf{dirx}, \mathsf{y}+\mathsf{diry})$, modelling the bird's movement along

Listing 2: Alignment.

```
1  agent Bird {
2      interface = . . .
3
4      Behaviour = Move; Behaviour
5      Move = {
6          p := pick 1;
7          dirx ← dirx_p;
8          diry ← diry_p;
9          x ← x + dirx;
10         y ← y + diry
11     }
12 }
```

its displacement vector (lines 10–11). We assume that the system evolves in discrete time steps and that at every step one bird executes one assignment.[3] Each assignment is performed atomically, but sequences of assignments may be susceptible to interleaving among different agents. To prevent this, i.e., to let the same agent execute multiple assignments atomically, these must be enclosed in curly brackets, as shown in lines 9–12.

Alignment. The specification above does not yield any form of collective behaviour, as the birds move independently of each other without considering their flockmates. Consequently, it becomes necessary to introduce models of birds that are influenced by the movements of their neighbours. In fact, flocking behaviour is commonly believed to be an outcome of local interaction mechanisms [22,42]. Specifically, we begin by examining *alignment*, which is the dynamic by which each bird modifies its heading based on the heading of its neighbouring birds. A trivial method to accomplish this is to allow each bird to mimic the direction of another bird in the flock. To model this behaviour, each bird must be able to observe the heading of other birds.

In Listing 2, lines 6–8, we present the modifications required to implement the alignment behaviour. To enhance clarity, we omit the interface as it is identical to that of Listing 1. It is worth noting that even though agents remain anonymous to each other, they possess a sense of *identity* provided by a unique *identifier* (id) assigned to each agent within the system. This identifier serves a similar purpose to the *this* or *self* keywords in general-purpose programming languages.

The availability of identifiers enables us to use the **pick** operator. It allows an agent to select another agent from the system in a non-deterministic manner. Specifically, in line 6, the instruction p := **pick** 1 selects the identifier of another agent and stores it in a *local* variable p. Generally, **pick** k returns k distinct identifiers that are guaranteed to be different from the identifier

[1] In this paper, we present condensed, human-readable versions of the full, machine-readable specifications. These are available at `https://github.com/labs-lang/labs-examples/tree/isola2022/isola2022`.

[2] Note that the specifications read $-D..D{+}1$ because ranges in LAbS are uppoer-bound exclusive (i.e., the value $D+1$ is *not* part of the range).

[3] We actually assume this for every LAbS model.

Listing 3: Cohesion.

```
1  agent Bird {
2      interface = ...
3
4      Behaviour = Move; Behaviour
5      Move = {
6          p := pick 1;
7
8          a_x := $x_p + \omega \cdot dirx_p$;
9          a_y := $y_p + \omega \cdot diry_p$;
10         sgn_x := 0 if x = a_x else
11                 −1 if x > a_x else 1;
12         diff_x := $d((x, 0), (a\_x, 0))$;
13         ... (Same for sgn_y, diff_y)
14         a_dirx := sgn_x · (D : 2 if diff_y > diff_x else D);
15         a_diry := sgn_y · (D : 2 if diff_y < diff_x else D);
16
17         dirx ← (dirx + a_dirx) : 2;
18         x ← x + dirx
19         ... (Same for diry, y)
20     }
21 }
```

Listing 4: Flock dispersion and birds collision.

```
1  agent Bird {
2      interface = ...
3
4      Behaviour = Move; Behaviour
5      Move = {
6          p := pick 1;
7          pIsIsolated := forall Bird b,
8              (b = p) or $d((x_p, y_p), (x_b, y_b)) > \delta$;
9          appId := id if pIsIsolated else p;
10
11         a_x := $x_{appId} + \omega \cdot dirx_{appId}$;
12         sgn_x := 0 if x = a_x else 1 if x > a_x else 1;
13         diff_x := $d((x, 0), (a\_x, 0))$;
14         ... (Same for a_y, sgn_y, diff_y)
15         a_dirx := sgn_x · (D : 2 if diff_y > diff_x else D);
16         a_diry := sgn_y · (D : 2 if diff_y < diff_x else D);
17
18         dirx ← (dirx + a_dirx) : 2;
19         diry ← (diry + a_diry) : 2;
20         posFree := forall Bird b,
21             ($x_b \neq x + dirx$) or ($y_b \neq y + diry$);
22         x ← x + dirx if posFree else x
23         y ← y + diry if posFree else y
24     }
25 }
```

of the agent performing the selection. We utilize the operator := to denote assignments to local variables, which are created implicitly upon their first assignment. In lines 7–8, we update the components of the heading vector by replacing them with the corresponding components of the selected agent. Since the bird now has the identifier of an agent stored in p, it can access the selected agent's heading vector using the syntax dirxp, diryp. In this specification, the bird updates its own heading vector with that of p (lines 7–8) and then moves by updating its own position (lines 9–10).

Cohesion. Birds in a real flock exhibit not only alignment in their direction of movement, but also cohesion, i.e., the ability to keep close to each other. However, the model of flocking seen so far does not accurately display this behaviour. In fact, even when two birds are far apart, they tend to move in the same direction without getting closer to each other. To address this limitation, we modify the current model to incorporate both alignment and cohesion. In our modified model, each bird selects another bird in the flock, estimates its future position based on its current direction, and then steers towards that position.

Listing 3 shows how *cohesion* can be modelled. We use the ternary operator a **if** c **else** b to represent a conditional expression that evaluates to a if condition c is true and b otherwise. The syntax $a : b$ denotes integer division with rounding, and $d((x_1, y_1), (x_2, y_2))$ denotes the Manhattan distance between two points, i.e., $|x_1 - x_2| + |y_1 - y_2|$. To implement the modified behaviour, each bird selects a target bird to approach (line 6). The agent estimates the target's position after ω steps (lines 8–9), and determines an approach vector (a_dirx, a_diry) pointing towards that position. The approach vector is computed component-wise (lines 11–15). The instructions for the y-component are omitted for sake of brevity. Finally, the bird's new heading vector is computed as the average of its current heading vector and the approach vector (line 17), providing the bird with some inertia for more realistic movement.

Avoiding flock dispersion and collisions. Despite the specifications provided so far, undesired outcomes such as flock dispersion and collisions can still occur. These problems arise when birds attempt to approach isolated birds or move to occupied locations, respectively. To avoid the former, we need to provide birds with the capability of checking whether a bird is isolated. Similarly, to prevent collisions, it is necessary for each bird to assess whether its target location is free of other birds before moving.

In Listing 4, we incorporate the above refinements. First, we check at line 7 whether a bird p is isolated, i.e., its distance from all other birds is greater than a parameter δ. To facilitate this check, we use quantified predicates that allow us to predicate over the attributes of all agents, or some agent, of given types. If the bird is isolated, it will continue along its current direction (line 9), and will not be approached by other birds. Similarly, to avoid collisions, a check at lines 20–23 is introduced to ensure that the bird only moves to an unoccupied position pointed at by its heading vector.

Listing 5: Fleeing from a predator.

```
1  agent Predator { ... }
2
3  agent Bird {
4      interface = . . .
5
6      Behaviour = Move; Behaviour
7      Move = {
8          p := pick 1 Bird;
9          . . .
10         a_diry := sgn_y · (D:2 if diff_y < diff_x else D);
11
12         e := pick 1 Predator;
13         e_x := x_e + ν · dirx_e;
14         esgn_x := 1 if x ≥ e_x else −1;
15         ediff_x := d((x, 0), (e_x, 0));
16         . . . (Same for e_y, esgn_y, ediff_y)
17         e_dirx := esgn_x·(D:2 if ediff_y > ediff_x else D);
18         e_diry := esgn_y·(D :2 if ediff_y < ediff_x else D);
19
20         e_dist := d((x, y), (e_x, e_y));
21         f_dirx := e_dirx if e_dist < λ else a_dirx;
22         dirx ← (dirx + f_dirx) : 2;
23         . . . (Same for f_diry, diry)
24         posFree := forall Bird b,
25             (x_b ≠ x + dirx) or (y_b ≠ y + diry);
26         x ← x + dirx if posFree else x
27     }
28 }
```

If another bird is already occupying that position, the bird stays in its current location.

Fleeing from a predator. Until now, we have considered a flock that is unperturbed by external threats. We shall now modify the model to enable birds to recognize predators and flee from them when they come too close, while preserving the flocking dynamics that we have introduced so far.

Listing 5 presents the modified implementation of the model. To implement this new behaviour, we make some slight modifications to the model. In particular, we refine the **pick** operator introduced earlier by making it typed. For example, at line 8, a bird selects another member of the flock and performs the same operations as seen in Listing 4. We omit some of the instructions for sake of brevity. Similarly, at line 12, the bird identifies a predator and evaluates its distance from itself. If the distance is too small, the bird will not approach it as usual, but instead flee from the predator. To model this fleeing behaviour, we compute a repulsive heading vector $(\mathsf{e\_dirx}, \mathsf{e\_diry})$ and let the bird follow it if the predator is closer than a given parameter λ.

LAbS agents and global knowledge. We should emphasize that certain operations in the specifications above (specifically, those at lines 6, 7–8, 11, and lines 20–21 of Listing 4) assume that a bird can access the state of other birds in the flock.

Listing 6: Description of the environment.

```
system {
    ...
    environment = ph[5] : 0
}
```

Formally, this requires each bird to possess global knowledge of the system; however, we argue that these operations are reasonable in practice by the fact that they model observations that a bird can carry out locally. For instance, in line 6, a bird can pick another flockmate by simply looking at its surroundings, without necessarily being aware of *every* bird in the system. Determining the direction of a bird (line 11), or whether it is isolated (lines 7–8), may be done by looking at that bird and the space around it. Lastly, finding out whether there is a bird at location $(\mathsf{x} + \mathsf{dirx}, \mathsf{y} + \mathsf{diry})$ (lines 20–21) does not require an interrogation of all birds in the system; rather, one can simply observe the location itself.

In conclusion, we have chosen to equip LAbS with these high-level primitives because they greatly simplify the specification process, and because they are usually *realistic*, i.e., they may be carried out by agents through local observations and without actual global knowledge. LAbS allows a rather high degree of freedom when using these primitives, but leaves to the user the problem of determining whether the corresponding observations are realistic or not.

2.2 Pathfinding ants

In this section, we set out to model an experiment [26] showing that a colony of ants is able to select the shortest path between two possible routes from its nest to a food source. Initially, ants distribute more or less evenly among the short and the long route. However, each ant releases a constant amount of pheromone as it walks from the nest to the food source and back. Each ant is also able to sense the concentration of pheromone present in the soil, and prefers the route where this concentration is higher. Since the shortest path gets quickly marked both by ants leaving the nest and by those returning with food, a feedback mechanism is triggered that makes all ants always select the shortest path, after a few minutes. The model ignores the fact that pheromone evaporates after a while, since the shortest path becomes established in a much shorter time span than the one after which evaporation becomes noticeable.

Description of the environment. We consider a simple scenario where the path diverges into two sub-paths

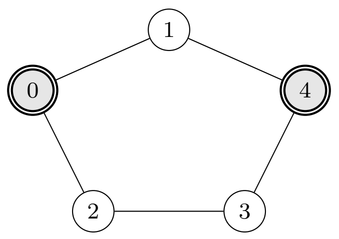


Fig. 2: Structure of the paths that ants can follow.

starting from the nest. The first sub-path, which is the shortest, can be traversed by an ant in one steps, whereas the second sub-path, which is the longest, requires each ant to take two steps. In the experiment proposed in [26], the ratio between long branches and short branches was indeed 2. Both sub-paths eventually converge where the food source is located. As an ant moves along either of the two sub-paths, it releases a certain quantity of pheromone with each step. It is important to note that, at the beginning of the system, the level of pheromone on both sub-paths is zero as no ants have passed through them yet.

The diagram of the model is depicted in Figure 2. We depict the ants' environment as a graph with five nodes, where the node labelled 0 corresponds to the nest and the node labelled 4 corresponds to the food source. As mention above, each node can hold a certain level of pheromone. We represent this phenomenon in Listing 6, by means of a shared array denoted as ph, initially set to zero.

Description of an ant. Each ant in the colony is characterized by two attributes, namely its *position* and its *direction*. All the ants in the colony start in the nest, and they must travel to the food source to obtain food. Once an ant leaves the nest, it must complete the entire journey to reach the food source before it can return to the nest. It cannot change direction halfway along the route.

Listing 7 shows a model for the above description. In particular lines 3–4 define the interface of an ant of the colony. Both attributes are represented using integer values. The attribute pos (line 3) identifies the position of an ant and corresponds to the number of graph node where it is located. This value is initialized to 0 as initially all the ants are in the nest. The attribute dir (line 4) indicates the direction of an ant. If it moves from the nest towards the food source, its value is 1, otherwise, it is -1. It is initially set to 1 as all the ants are initially located in the nest.

Behaviour of an ant. An ant located in the nest chooses, if possible, the path with the highest level of pheromone. If the pheromone levels of both paths are similar, the ant non-deterministically selects one of them. Upon selecting a path, the ant releases a specific amount of pheromone along that path. When the ant reaches the food source, it again chooses its path based on the level of pheromone present. It is worth noting that the ant can return along a different route than the one it used on its outward journey.

Listing 7: Description and behaviour of an Ant.

```
 1 agent Ant {
 2     interface =
 3         pos: 0;
 4         dir: 1
 5
 6     Behavior = (Next0
 7                 ++
 8                 Next4
 9                 ++
10                 NextOther);
11                DropPh; Behaviour
12
13     Next0 = pos = 0 →
14         (ph[1] − ph[2] ≤ δ) → (pos, dir ← 2, 1)
15         ++
16         (ph[2] − ph[1] ≤ δ) → (pos, dir ← 1, 1)
17
18     Next4 = ...
19
20     NextOther = pos ≠ 0 ∧ pos ≠ 4 →
21         pos ← (4 if dir = 1 else 0) if pos = 1 else
22               (3 if dir = 1 else 0) if pos = 2 else
23               (4 if dir = 1 else 2) if pos = 3 else −1
24
25     DropPh = {
26         ph[pos] ⇐ ph[pos] if (pos = 0 ∨ pos = 4)
27                            else ph[pos] + 1;
28     }
29 }
```

Listing 7 shows the implementation of the behaviour above. The behaviour is defined recursively in lines 6–11. The operator ++ denotes a nondeterministic choice between behaviours. In our case, the agent has three options, listed in lines 6–10 and named **Next0**, **Next4**, and **NextOther**. Their definition is at lines 14–16. Each of these processes is actually guarded by the ant's current position: for instance, **Next0** can only proceed when the ant is at node 0 (i.e., the nest). In this case, the ant updates its position based on the concentration of pheromone in each potential destination. If the two concentrations are different enough (i.e., one is at least δ units more than the other, where δ is an external parameter)the ant will always move towards the higher concentration. Otherwise, the ant nondeterministically selects one of the two nodes 1 and 2. We omit the definition of **Next4** for the sake of brevity, as it is similar to that of **Next0**. **NextOther**, on the other hand, is activated when an agent is in positions 1, 2, or 3 (i.e.

Listing 8: Baseline agent modelling.

```
1  agent Ant {
2      interface =
3          pos: $0..\lambda$;
4          dir: $[-1, 1]$
5
6      Behavior = Move; Behaviour
7      Move = {
8          desPos := pos + dir;
9
10         pos ← desPos if $1 \leq$ desPos $\leq \lambda$ else
11                  $-1$ if desPos $< 0$ else $\lambda + 2$
12     }
13 }
```

Listing 9: Collision.

```
1  system {
2      ...
3      environment = collide[$\lambda$] : 0; bar[$\lambda$] : $0..N$
4  }
5
6  agent Ant {
7      interface =  . . .
8
9      Behavior = Move; Behaviour
10     Move = {
11         dir ← −dir if collide[pos] = 1 else dir;
12         collide[pos] ⇐ 0;
13
14         desPos := pos + dir;
15         dirDpos := 0 if bar[desPos] = −1
16                    else $\text{dir}_{\text{bar[desPos]}}$;
17         collide[desPos] ⇐ 0 if dirDpos = dir else 1;
18
19         bar[pos] ⇐ −1;
20         dir  ←  0 if (desPos < 1 ∨ desPos > $\lambda$) else
21               dir if dirDpos = dir else  − dir;
22         pos ←     0 if desPos < 1 else
23              $\lambda + 2$ if desPos > $\lambda$ else
24             desPos if bar[desPos] = −1 else pos;
25         bar[pos] ⇐ id
26     }
27 }
```

along one of the paths, and not at the nest or food source). In this case, the agent simply continues along its direction with no possibility of reversal.

After moving to another node, the agent drops a unit of pheromone at the new location, as shown in line 27. Finally, the ant repeats its behaviour from the beginning.

2.3 Ants on a bar

In this Section we describe the example proposed by Fettke at al. in [23], where a group of ants moves back and forth on a one-dimensional bar.

Description of an ant. Each ant of the group may be characterised by two distinct attributes: its *position* and its *direction*. The former is denoted by a coordinate pos, which represents the ant's location along the bar. The latter is described using an integer dir.

In Listing 8 (lines 2–4), these attributes are initialised non-deterministically. Specifically, each ant may assume any position on a bar with a length of λ, i.e. from 1 to λ inclusive. Additionally, the direction attribute is initialised using a random selection from one of two possible values: 1, indicating that the ant will move to the right, or -1, indicating that the ant will move to the left.

Behaviour of an ant. We initially focus on a straightforward system in which an ant moves in accordance with its direction and falls off the edge when it reaches it. Several ants can currently move side by side.

In Listing 8, lines 7–12 describes this simple behaviour. Please note that it is defined again recursively as shown in line 6. It outlines that each agent continuously executes the actions defined in the **Move** process. More comprehensively, an agent initially considers the desired position to which it will move, dependent upon its current location and direction. The process for updating this position is presented in lines 10–11. It should be noted that if an agent continues to move along the bar, there are no issues (line 10). However, if an agent reaches the left-hand edge, it is relocated to position -1, which is outside the bar, while if an agent reaches the right-hand edge, it is positioned at position $\lambda + 2$, which is again outside the bar (line 11).

Collisions. Let us now refine the description of ant behaviour to obtain the one presented in the original example [23]. An ant, as before, can move forward or backward along the bar. Two ants walking towards each other, will eventually collide. Upon collision, both ants reverse their direction and continue along the bar. It is important to note that ants cannot share the same position or overtake other ants. Finally, when an ant reaches the edge, it falls.

Listing 9 presents the behaviour described above. It contains several sections. In lines 1–4 we define a shared array collide, initialised with zeroes, which enables the signal of a collision to travel from one agent to another. Additionally, we initialize a shared array called bar, which stores the ids of the agents based on their position on the bar. The behaviour of an ant agent is described in lines 9–26. It is defined recursively, with each agent repeatedly performing the **Move** process. The latter can be divided into three parts. In the first part (lines 11–12), the ant checks whether another one has previously collided with it by examining whether the collide array is set to 1. If a collision has occurred, the ant immediately changes its direction and then removes the occurrence from the array of collisions by setting it back to 0. It is important to note that the assignments to shared variables are denoted by the symbol $\Leftarrow$ to distinguish them from other kinds of assignment seen so far. In the second part (lines 14–17), the agent checks whether the position of the bar it wishes to move to is already occupied. If it is, the agent examines the direction of the agent occupying that spot (line 16). Moreover, if the position is occupied and the agent is moving in the opposite direction, it records a collision occurrence in the collide array. In the third and final part (lines 19–25), the agent first removes its own position on the bar. Then, the direction and position are updated, taking into consideration both the case where an ant is on the edge of the bar and the case where a collision has occurred. Finally, the new position on the bar is updated.

Listing 10: Constraints.

```
1  assume {
2      GridCentre = forall Bird b,
3          x_b > 490 and x_b ≤ 510 and
4          y_b > 490 and y_b ≤ 510
5      DifferentPositions = forall Bird a, forall Bird b,
6          a = b or x_a ≠ x_b or y_a ≠ y_b2
7      DirectionNotNull = forall Bird b,
8          dirx_b ≠ 0 or diry_b ≠ 0
9  }
```

Listing 11: Predator specifications.

```
1  agent Predator {
2      interface =
3          x: 480;
4          y: 480;
5          dirx: 3;
6          diry: 3
7
8      Behaviour = Move; Behaviour
9      Move = {
10         x ← x + dirx;
11         y ← y + diry
12     }
13 }
```

3 Analysis results

In this section, we analyse the three examples described in Section 2, both exhaustively and through simulation. We performed all the experiments in a virtualized environment on a dedicated machine running 64-bit GNU/Linux with kernel 5.4.0 and equipped with four 2-GHz Xeon E7-4830v4 10-core processors and 512 GB of physical memory.

3.1 Flock of birds

In this subsection, we investigate whether the current specifications of the flock model enable it to maintain its cohesion when threatened by a predator. To test this, we set up an experiment where all birds start from random positions in a small area, and a predator bird flies through the centre of this area, posing a threat to the flock. Our objective is to demonstrate that the predator's attack disrupts the cohesion of the flock, causing it to scatter. However, we also aim to show that the flock can effectively reorganize itself and regroup once the predator leaves, demonstrating the effectiveness of the model in mimicking complex emergent behaviour.

In our experiment, we consider an *arena* modelled as a 1024×1024 square, within which all agents are placed. However, if the birds were allowed to assume any position within the arena, they could be widely scattered. Therefore, we initialize the flock with the birds starting close to each other, which is a more realistic starting position for an unperturbed flock. To ensure diversity, we also prevent the birds from starting at the same position as others or from being stationary, which means having a null heading vector. By imposing these constraints, we can ensure that our simulation begins with a realistic initial configuration, which closely approximates the behaviour of a real flock of birds.

Listing 10 shows how to model these initial constraints. These are listed in a new section of the specifications titled **assume**, where each constraint is expressed through a quantified predicate, as seen in Section 2. Lines 2–4 establish that birds can only be placed in a 20×20 sub-grid at the centre of the arena, which guarantees that they will not be too far apart. It is worth noting that the flock will never reach the edges of the arena due to this initial configuration and the limited number of steps that will be analysed. To prevent two agents from starting at the same position, lines 5–6 state that two agents cannot assume the same initial position. Lastly, line 7 prescribes that every bird must have a non-null heading vector.

The predator agent is specified in Listing 11. The predator features the same attributes as the birds in the flock, i.e., a position (x, y) and a heading vector $(\mathsf{dirx}, \mathsf{diry})$. To ensure that the predator intersects the flock, it is given a very simple behaviour such that it moves in a straight line along its initial heading vector. The initial position and the heading vector of the predator are given determined values in lines 3–6. The predator is given a longer heading vector than those of flock birds to model the fact that it moves faster. The predator's behaviour is shown at lines 8–12 and is ex-

Listing 12: Specifying a cohesion requirement.

```
check {
    Cohesion = after $B$ forall Bird a, forall Bird b,
        a = b or $d((x_a, y_a), (x_b, y_b)) \leq k$
}
```

actly like the one seen in Listing 1, modelling movement in a straight line.

Our goal here is to investigate the cohesion of a flock after a predator attack. Specifically, we aim to examine whether the distance between birds increases as they flee from the predator and whether this distance returns to its previous level after the predator departs. This property is formalized in the **check** section of our specifications, as illustrated in Listing 12. The property is described in line 3, where the **after** B modality indicates that the maximum distance between any two birds should not exceed a given parameter k, B steps after the initial state. In LTL [39], this construct can be expressed as a sequence of B "next" operators denoted as X^B.

In order to assess the ability of our flock to exhibit the desired behaviour, we employed a simulation workflow (see Fig. 3) that quickly generates random traces of our specification. The workflow was implemented in SLiVER,[4] a language originally aimed at formal verification of collective adaptive systems [16,17].

Our approach involves encoding the specifications into a sequential imperative program [16]. Tools for reachability analysis are then used to generate one or more random traces of a desired length. More specifically, a simulation of length B for a given system can be obtained by encoding the system as a program that keeps track of the number of executed system steps in a specific variable `steps`, and then by checking that all reachable states have `steps` $< B$.[5] The main issue with this approach is that most reachability tools are deterministic, so we would always obtain the same simulation. To address this issue, we use as our back end a SAT-based bounded model checker [10] together with a randomized solver, so that the same reachability query may produce different counterexamples (each corresponding to a feasible simulation of our system). These traces are subsequently translated into the specification syntax and presented to the user. Also, these traces provide valuable information about the satisfaction of properties within the specification.

In order to enhance the efficiency of our simulation workflow, we use a concretization step that is performed prior to feeding the program to the back end. This steps randomly picks feasible concrete values for a number of symbolic variables in the emulation program (e.g., the ones representing the initial state of the system or the choices made by the scheduler), and then preloads these values into the back end as *weak* assumptions. The back end will try to honour all weak assumptions, but it is free to drop one or more if the reachability query would be otherwise unsatisfiable. This approach seems effective in speeding up the back end.

The parameters and values used in our models and simulations, as well as the composition of the system, are summarized in Table 1. Notably, we use B both as the bound of the cohesion property and as the desired length of our simulations. Our simulations assume round-robin scheduling, meaning that each trace consists of a sequence of *epochs* in which each agent performs exactly one action. It is important to note that in this context, an atomic block is considered a single action. While this assumption is demanding, we find it reasonable when modeling real-world systems. Furthermore, it is much weaker than the implicit synchrony assumptions made in other models, such as those found in [42,2]. These models require all agents to evolve in lockstep, meaning that the future state of individual agents depends on the current state of the entire system, and state changes happen simultaneously for all agents.

Table 1: Parameters in the model presented in Listing 5 and their values used in the simulation process.

Name	Description	Value
B	Bound for the cohesion property	600
D	Maximal absolute value of heading vector components for birds	2
G	Size of the arena	1024
δ	Isolation distance	32
λ	Safe distance from predator	32
ν	Used to estimate the future position of the predator	2
ω	Used to estimate the future position of the bird to approach	14
k	Maximal distance to satisfy the cohesion property	$0 - 40$
	Number of Bird agents	29
	Number of Predator agents	1

The experimental outcomes are depicted in Fig. 4. The x-axis of the plot denotes the cohesion coefficient, k, which was varied from 0 to 40, representing the maximum separation distance between the two elements in their initial configuration. The y-axis represents the percentage of traces that were found to satisfy the cohesion property. Each data point on the plot corresponds

[4] `https://github.com/labs-lang/sliver/`

[5] Of course, here we are assuming that the system does not always deadlock in less than B steps.

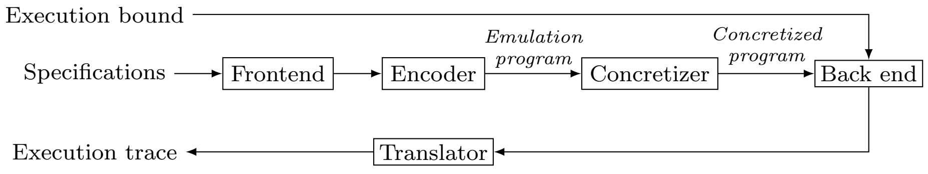


Fig. 3: Workflow to simulate our specifications.

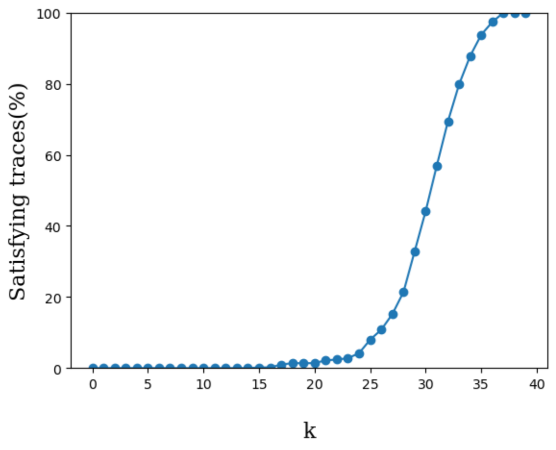


Fig. 4: Percentage of traces satisfying the property in Listing 12 as k varies.

to 1000 simulations. In the graph, a sharp decline in the number of traces that satisfy the property is observed.

The visual representation of a trace generated by our simulation process is presented in Fig. 5. The birds are depicted by triangles that point towards the direction of their heading vector, and the predator is depicted as a larger, red triangle with a black outline. It is important to note that in this trace, the birds are never in the same position, and any overlapping triangles are a visual artefact. As expected, the trace demonstrates that the predator's attack causes some dispersion in the flock as the birds attempt to evade the threat. However, the birds are eventually able to regroup and reorient themselves coherently, thereby satisfying the property we specified in Listing 12. It is worth mentioning that our simulation workflow proved useful during the specification process as it helped us realize the potential for flock dispersion in Listing 3, which led to the development of the more refined Listing 4.

Independent replication. The main findings about the collective behaviour of this model have been replicated [44] by reimplementing the model with Python and the Mesa agent-based modelling library.[6] The individual behaviour for flocking birds and the predator have been manually translated from LAbS to Python; then, 10 simulations have been run featuring 600 epochs, 100 flocking birds, and 1 predator. All simulations show that the flock is able to reattain cohesion after being disrupted by the bird of prey.

[6] `https://github.com/projectmesa/mesa/`

Listing 13: Specifying the shortest path property.

```
1 check {
2     Shortpath = after B forall Ant a,
3         pos_a = 0 ∨ pos_a = 1 ∨ pos_a = 4
4 }
```

3.2 Pathfinding ants

In this Section, we investigate the path choices of an ant colony during food search and transport.

In particular, our goal is to show that if the ants have two paths of different lengths available that lead from the nest to the food, they tend to choose the shortest path and gradually abandon the longer one. More in detail, we want to show that after a certain amount of time has passed, the ants will be located exclusively in the nest, on the shortest path, or at the food source. This property is formalized in the **check** section of the specifications and is illustrated in Listing 13. The property described in lines 2–3 asserts that, **after** B steps from the initial state, the nodes where any ant is located are 0, the nest, 1, the shortest path, or 4, the location of the food.

The exhaustive verification of the described property against such a system gives a negative result. A trivial counterexample is given by the case in which each ant in the system initially chooses the longest path to explore. In this case, the amount of pheromone on the short path will remain zero. Once the ants have to choose the path for the return, none of them will consider the short path. However, this counterexample appears to be unrealistic. As demonstrated in numerous experiments [26,21,19], ants initially tend to distribute themselves evenly along each branch and do not all cluster on a single branch.

The experimental setup we present involves simulating the system a certain number of times, extracting

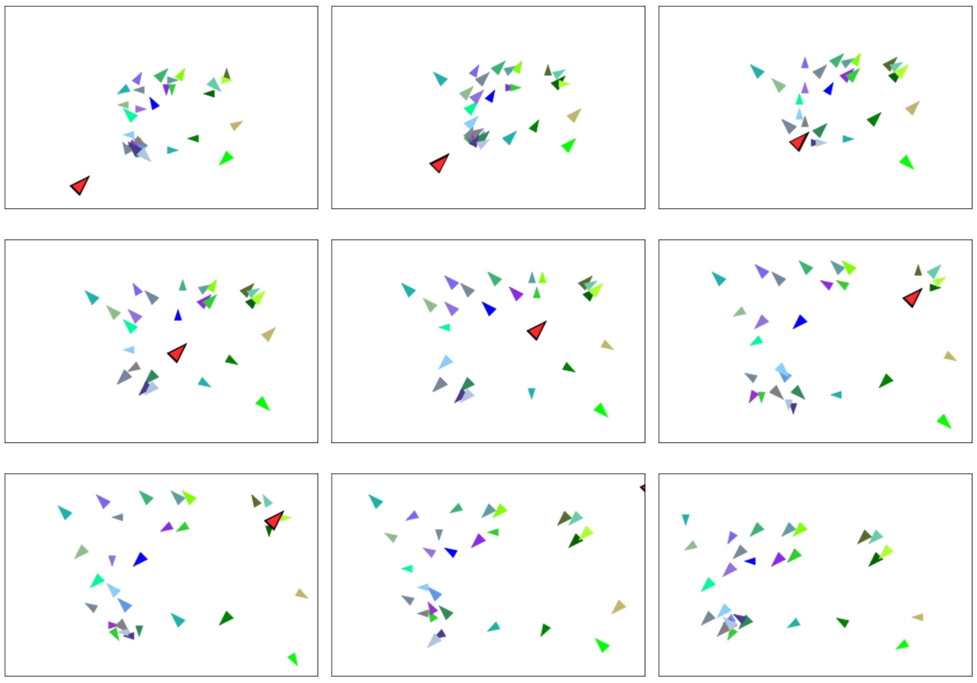


Fig. 5: A trace generated through simulation. The predator is the red triangle with black outline.

Table 2: Parameters in the model described in Listing 7 and their values.

Name	Description	Value
N	Number of Ant agents	10, 15, 20, 25
δ	Pheromone difference threshold	**N**, **N**/2, **N**/4
γ	Quantity of pheromone dropped	1

traces, and checking whether each of them satisfies the given property or not.

The simulation process resembles that presented in Section 3.1 and it is shown in Figure 3. The only notable difference is that the initial concretization step, in which each non-deterministic assignment is replaced by a randomly selected concretized value, no longer covers the choice of the initial state, as this is now deterministic. The concretization process is now applied to the choices made by the scheduler and to the internal choices of each agent's behaviour, i.e. when it must choose one of the two paths to follow and both can be selected.

In Table 2, we show the parameters used during the simulation phase. In Table 3 we present the results obtained. For each combination of the parameters, we perform 1000 simulations. Note that the parameter γ, which we set to 1 for each simulation, does not appear among the parameters in Table 3. In Figure 6, we represent the results graphically. As expected, as the

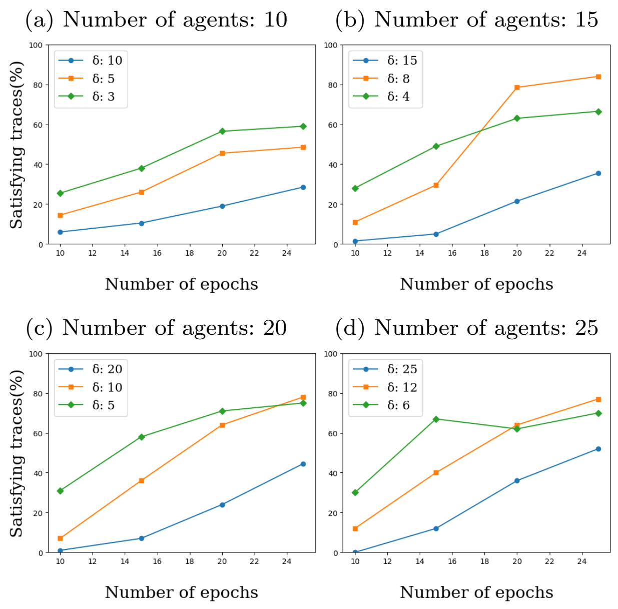


Fig. 6: Percentage of simulations that satisfied the property in Listing 13.

number of epochs increases, an increasingly number of simulations satisfy the described property. Note that, despite the increasing trend for each system and each δ considered, the growth is faster when δ is equal to half of the agents in the system. Since the ants dis-

Table 3: Simulation results for the pathfinding ants system. **B** is the number of epochs. **Sat** is the percentage of simulations in which the property is satisfied.

(a) $\mathbf{N} = 10$.

δ	**B**	**Sat** (%)
10	10	6.0
10	15	10.5
10	20	19.0
10	25	28.5
5	10	14.5
5	15	26.0
5	20	45.5
5	25	48.5
3	10	25.5
3	15	38.0
3	20	56.5
3	25	59.0

(b) $\mathbf{N} = 15$.

δ	**B**	**Sat** (%)
15	10	1.5
15	15	5.0
15	20	21.5
15	25	35.5
8	10	11.0
8	15	29.5
8	20	78.5
8	25	84.0
4	10	28.0
4	15	49.0
4	20	63.0
4	25	66.5

(c) $\mathbf{N} = 20$.

δ	**B**	**Sat** (%)
20	10	1.0
20	15	7.0
20	20	24.0
20	25	44.5
10	10	7.0
10	15	36.0
10	20	64.0
10	25	78.0
5	10	31.0
5	15	58.0
5	20	71.0
5	25	75.0

(d) $\mathbf{N} = 25$.

δ	**B**	**Sat** (%)
25	10	0.0
25	15	12.0
25	20	36.0
25	25	52.0
12	10	12.0
12	15	40.0
12	20	64.0
12	25	77.0
6	10	30.0
6	15	67.0
6	20	62.0
6	25	70.0

tribute themselves uniformly along each possible path and since they have to choose between two paths, it seems reasonable that the presence of half of the colony on one of the paths is a reasonable threshold to determine which path to follow.

In conclusion, the visualization of a trace generated by the simulation process is shown in Fig. 7. The node on the left represents the nest, while the one on the right represents the food source. The top path is the shortest and can be covered in a single movement, while the bottom path represents the longer path that requires two movements to be covered. The ratio between the lengths is therefore 2. As expected, the trace shows that, starting from an initial state where the level of pheromone on both paths is zero, the ants initially distribute themselves almost uniformly on both paths. However, after a certain amount of time, the colony begins to show a preference for the shortest path, until the choice becomes dominant. This behaviour is represented by the colour gradient, which expresses the amount of pheromone present on the path segment.

Table 4: Parameters in our model and their values.

Name	**Description**	**Value**
λ	Length of the bar	12
	Number of Ant agents	6

Listing 14: Constraints.

```
1  assume {
2      Position = forall Ant a, forall Ant b,
3                  (id_a < id_b ∨ pos_a ≥ pos_b) ∧
4                  (id_a ≥ id_b ∨ pos_a < pos_b)
5
6      Direction0 = exists Ant a, id_a = 0 ∧ dir_a = 1
7      Direction1 = exists Ant a, id_a = 1 ∧ dir_a = −1
8      Direction2 = exists Ant a, id_a = 2 ∧ dir_a = 1
9      Direction3 = exists Ant a, id_a = 3 ∧ dir_a = 1
10     Direction4 = exists Ant a, id_a = 4 ∧ dir_a = −1
11     Direction5 = exists Ant a, id_a = 5 ∧ dir_a = 1
12
13     Bar = forall Ant a, (pos_a = bar[1] ∧ bar[1] = id_a) ∨
14                         (pos_a ≠ bar[1] ∧ bar[1] = −1)
15                         ...
16                         (pos_a = bar[λ] ∧ bar[λ] = id_a) ∨
17                         (pos_a ≠ bar[λ] ∧ bar[λ] = −1)
18 }
```

3.3 Ants on a bar

In this section, we illustrate how exhaustive verification can be leveraged to reason about the event ordering of a concurrent system. By using the example proposed by Fettke et al. in [23], our goal is to demonstrate how it is feasible to construct a directed graph that describes the (partial) order of ant collisions and bar falls, using the agent behaviour specifications.

We define the initial state of the system as described in [23]. Table 4 presents the parameters utilized during the verification phase. Additionally, we impose the initial conditions presented in Table 14. These ensure that the ants are positioned exactly as in the considered initial state. Specifically, in lines 2–4, we enforce that the ants are directed according to their implicitly assigned id. In lines 6–11, we assign the initial directions, which again resemble those of the system presented in the paper. Finally, in lines 13–17, we initialize each position of the bar with the id of the ant located on it.

We now demonstrate how to save a collision or a fall event of an ant from the bar. To achieve this, it is necessary to modify the specifications of the system and the ants, which are provided in Listing 9 in Section 2. It is worth noting that the modifications we introduce do not affect the behaviour of an ant, but are only for event logging purposes.

The updated specifications are shown in Listing 15. In detail, we define a shared array that will contain the events and a counter that will allow us to write to the

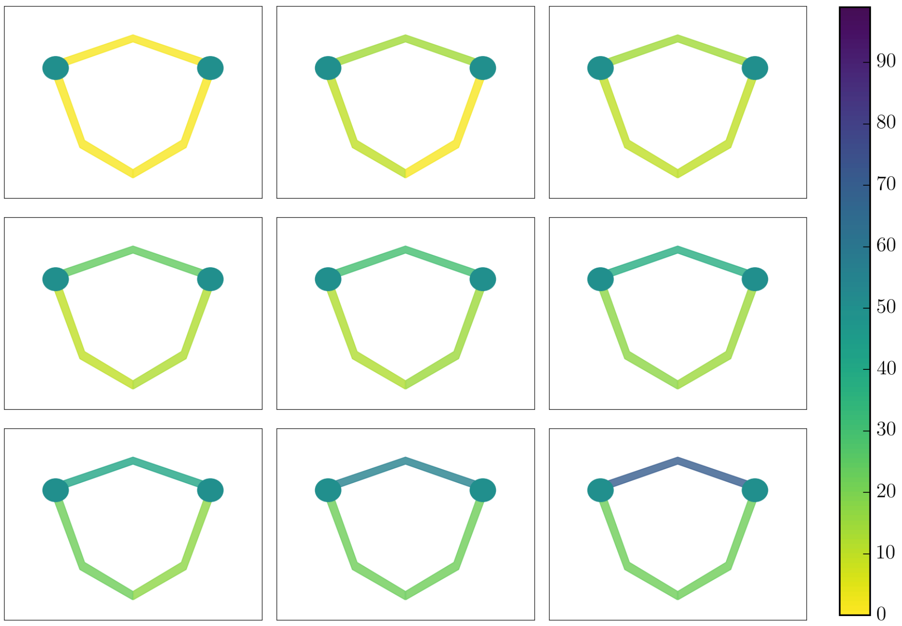


Fig. 7: Visualization of a simulated trace for the pathfinding ants system. The circles represent the nest (on the left) and the food source (on the right). The upper branch represents the shortest path, i.e. node 1 of Fig. 2. The lower branch is divided into two identical sections that respectively represent nodes 3 and 4 of Fig. 2. Colours denote the concentration of pheromone on each path.

Listing 15: Events logs.

```
1  system {
2      ...
3      environment = events[100] : 0; counter : 0
4  }
5
6  agent Ant {
7      interface =  ...
8
9      Behavior = Move; Behaviour
10     Move = {...
11         events[counter] ⇐
12          id · 100 if desPos = 0 ∨ desPos = λ + 1 else
13          (id − 1) · 10 + id if dir = −1 ∧ dirDpos = 1 else
14           id·10+(id+1) if dir = 1 ∧ dirDpos = −1 else
15         −1
16
17         counter ⇐ counter if events[counter] = −1 else
18                 counter + 1
19         ...
20     }
21 }
```

Listing 16: Specifying the consequences properties.

```
1 check {
2     Fallen = after B forall Ant a, pos_a < 1 ∨ pos_a > λ
3
4     Consequence = after B $consequence(E_1, E_2)
5 }
```

array (line 3). The array is initialized with the value −1. Before updating their direction and position, each agent, if necessary, records an event. In line 12, a fall from the bar by one of the agents is recorded. This event is coded as a multiple of 100. For example, the value 200 indicates that the agent with id 2 has fallen. In line 13, a collision is recorded. The id of the agent on the left is found in the tens place, while the id of the agent on the right is found in the ones place. For example, the event 45 corresponds to a collision between agents 4 and 5. In all other cases, the value of the array is not updated. Finally, the counter is only incremented if the value has been modified.

We now focus on verifying the ordering of events in our system. In order to do so, we first need to know the number of steps after which all ants inevitably fall off the bar (so, no other collision is possible after that). It is clear that such a value should exist, but it is not known in advance; however, we van find it by applying formal verification to our system. However, we can find this value by applying formal verification. To do so, we formalize the property "after B steps, every ant has fallen from the bar" (Listing 16, line 2). Then, we verify this property several times with increasing values of B, until it is verified. Once we obtain a successful outcome for some value B^*, we know that all events take place in the first B^* steps of every execution.

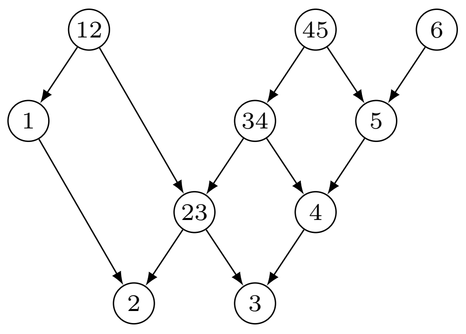


Fig. 8: The events with a single digit denote that the agent with the corresponding id has fallen from the bar. The events with two digits are collisions between the agents with the corresponding ids.

Now, we can check if a given event E_1 is inevitably preceded by another one E_2 (Listing 16, line 4).[7]

By verifying all possible combinations of all events while using a verification bound at least as great as B^*, it is possible to construct a directed graph that describes the (partial) order of collisions and falls. If it is verified that event E_1 is a consequence of event E_2, then in the event graph, E_1 precedes E_2. If, on the other hand, this is not verified, the order of the two events is inverted. If the reversed order is verified E_1 follows E_2 in the event graph, otherwise, it means that the two events are not comparable and do not depend on each other. We performed 23 verification tasks, and obtained the graph shown in Fig. 8. This graph closely replicates the ordering of events outlined in the original presentation of the system [23, Fig. 7], but interestingly we obtain it in a fully automated fashion, rather than by manually reasoning on the specifications.

4 Related work

Modelling of flocking behaviours in the literature relies on different approaches, including equational modelling through techniques such as differential equations [52], discrete-time dynamics [2,42], or statistical mechanics [6]; decentralized control laws, either defined adhoc [54] or synthesized from a centralized controller [34]; or language-based specifications, as presented in this work. The behaviour of ant colonies has also been studied following different approaches. For trail foraging, techniques such as modelling through nonlinear differential equations [48,4] or practical experimentation [5, 26] have been used.

Language-based approaches offer the advantage of facilitating the refinement and comparison of models with minimal effort. For instance, the framework proposed in [31] has been used to model various predator tactics and versions of flocking behaviour. Simulations have shown that flocks with more social tendencies exhibit better survival rates, whereas those with individualistic tendencies are more vulnerable to predation [13]. Regarding ant colonies, the Weighted Synchronous Calculus of Communicating Systems (WSCCS) has been utilized to model their activities, such as the sorting of the brood pile [50] and task allocation [51]. It has also been used to demonstrate how a colony responds to external disturbances [49]. StarLogo [41] and NetLogo [55] are further examples of language-based, bottom-up modelling frameworks; compared to our platform, they are more oriented towards analysis through massive simulations, as they can handle hundreds of agents with ease. NetLogo simulations can even become interactive by defining user interface controls to dynamically alter model parameters as the simulation runs. However, we are not aware of any work that applies formal verification to StarLogo or NetLogo models: such an effort would likely be hindered by a lack of formal semantics for either language and by their rather dynamic nature.

Formal specification languages also enable exhaustive exploration of the state space, which may provide strong guarantees on the behaviour of a system or detect subtle bugs that may be difficult to detect through simulations alone. For example, the alpha algorithm [56], designed to induce a flock of dispersed agents to aggregate in a small region of space, was found to be incorrect [29,1] by verifying models of the algorithm written in ISPL [33] or NuSMV [9]. Emulation programs may similarly enable formal analysis of high-level specifications by means of structural encodings towards lower-level languages, allowing for the reuse of different existing verification technologies [16,18].

Bottom-up and simulation-aided design is also commonplace in the engineering of robot swarms and similar classes of robotic systems [7]. In this context, robots are typically programmed individually using general-purpose languages like C++ or Python or higher-level, domain-specific, formalisms [15,37], with possible reliance on existing robotic middleware like ROS [40]. The resulting programs are evaluated by simulating the robots under one of several available simulation platforms [28,38,43] to empirically assess whether the swarm exhibits adequate collective behaviour.

[7] The `$consequence(...)` syntax means that this check is currently implemented as an external function written in C, which compares the array indices at which E_1 and E_2 appear. We plan to turn this into a feature of our property language in an upcoming release of SLiVER.

5 Conclusion

This work has shown how compositional models can help to reason about the individual dynamics that lead to emergent behaviour of collectives. We presented three models that describe the behaviour of natural systems in a step-by-step, yet intuitive and concise manner.

By using an automated simulation workflow, we have shown that birds split into smaller groups to avoid the threat, before reassembling once the danger subsided. This behavior has been observed in both real-life flocks and in other models [2]. Using the same workflow, we demonstrated how an ant colony is capable of selecting the shortest path to reach a food resource. Also this behavior has been previously observed both in real-life scenarios and in other models [19, 3, 20]. Finally, through comprehensive verification, we showed the ability to reason about the order of events in a very simple system considered in a paper presented at the conference to which this special issue is dedicated.

In light of the promising results presented in this paper, there are several avenues for future research in the field of compositional modeling of natural collective behaviors. Firstly, it is important to further develop and refine the simulation workflow used in this study, as it is still in the experimental phase. Indeed, although the workflow was successful in simulating the model presented in Section 2, its limitations in handling specifications that contain guarded statements need to be addressed. These scenarios may prove difficult to simulate, as certain concretizations may fail to satisfy certain guards, making it impossible to generate traces of the desired length.

To address the challenges we plan to adapt our backend tool to enable the modification of concretization constraints until a valid trace is obtained.

We plan also to enhance the simulation-based approach by incorporating exhaustive state space exploration techniques. This complementary method may formally prove the emergence of expected collective behaviors, regardless of the initial state or the specific interactions between agents within the system. Moreover, to achieve our goal of formal verification of the emergence of desired collective behaviors, we propose to adapt existing techniques based on verification of emulation programs [16]. This adaptation may involve extending these techniques to support expressive temporal logics such as LTL [39]. To achieve this, a more rigorous formalization of the linguistic constructs introduced in Section 2 may also be necessary.

Given the cost of exhaustive analysis for large systems, we also plan to extend our simulation workflow to enable lightweight formal methods, such as statistical model checking [45]. This approach will enable us to obtain statistical evidence on the correctness of the system. The ability of our framework to verify property satisfaction relying on simulation can be considered a rudimentary form of runtime verification [32]. We plan to extend this capability to include larger classes of monitorable properties [25].

Parallelization of our simulation workflow can be easily achieved by distributing the workload among multiple machines, and we plan to investigate the possibility of implementing distributed techniques in the back end to further improve performance [27]. These efforts will allow us to generate a large number of traces for massive systems. Effective visualization of the textual traces is also crucial for supporting the design process. While our current automated visualization tool works well for the flocking case study (as demonstrated in Fig. 5), we aim to build a more generic framework or integrate our workflow into existing simulation platforms to provide a more flexible and versatile visualization tool.

References

1. Antuña, L.R., Araiza-Illan, D., Campos, S., Eder, K.: Symmetry reduction enables model checking of more complex emergent behaviours of swarm navigation algorithms. In: 16th Annual Conference Towards Autonomous Robotic Systems (TAROS), *LNCS*, vol. 9287, pp. 26–37. Springer (2015). DOI 10.1007/978-3-319-22416-9_4
2. Ballerini, M., Cabibbo, N., Candelier, R., Cavagna, A., Cisbani, E., Giardina, I., Lecomte, V., Orlandi, A., Parisi, G., Procaccini, A., Viale, M., Zdravkovic, V.: Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study. Proceedings of the National Academy of Sciences **105**(4), 1232–1237 (2008). DOI 10.1073/pnas.0711437105
3. Beckers, R., Deneubourg, J.L., Goss, S.: Trail laying behaviour during food recruitment in the ant lasius niger (l.). Insectes Sociaux **39**, 59–72 (1992)
4. Beckers, R., Deneubourg, J.L., Goss, S., Pasteels, J.M., et al.: Collective decision making through food recruitment. Insectes sociaux **37**(3), 258–267 (1990)
5. Bernstein, R.A.: Foraging strategies of ants in response to variable food density. Ecology **56**(1), 213–219 (1975)
6. Bialek, W., Cavagna, A., Giardina, I., Mora, T., Silvestri, E., Viale, M., Walczak, A.M.: Statistical mechanics for natural flocks of birds. Proceedings of the National Academy of Sciences **109**(13), 4786–4791 (2012)
7. Brambilla, M., Ferrante, E., Birattari, M., Dorigo, M.: Swarm robotics: A review from the swarm engineering perspective. Swarm Intelligence **7**(1), 1–41 (2013). DOI 10.1007/s11721-012-0075-2
8. Cederman, L.E.: Endogenizing geopolitical boundaries with agent-based modeling. Proceedings of the National Academy of Sciences **99 Suppl 3**, 7296–7303 (2002). DOI 10.1073/pnas.082081099
9. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella,

A.: NuSMV 2: An opensource tool for symbolic model checking. In: 14th International Conference on Computer Aided Verification (CAV), *LNCS*, vol. 2404, pp. 359–364. Springer (2002). DOI 10.1007/3-540-45657-0_29
10. Clarke, E., Kroening, D., Lerda, F.: A tool for checking ANSI-C programs. In: 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), LNCS, pp. 168–176. Springer (2004). DOI 10.1007/978-3-540-24730-2_15
11. De Nicola, R., Di Stefano, L., Inverso, O.: Multi-agent systems with virtual stigmergy. Science of Computer Programming **187**, 102345 (2020). DOI 10.1016/j.scico.2019.102345
12. De Nicola, R., Di Stefano, L., Inverso, O., Valiani, S.: Modelling flocks of birds from the bottom up. In: T. Margaria, B. Steffen (eds.) 11th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Adaptation and Learning (ISoLA), *LNCS*, vol. 13703, pp. 82–96. Springer (2022). DOI 10.1007/978-3-031-19759-8_6
13. Demsar, J., Lebar Bajec, I.: Simulated predator attacks on flocks: A comparison of tactics. Artificial Life **20**(3), 343–359 (2014). DOI 10.1162/ARTL_a_00135
14. Deneubourg, J.L., Pasteels, J.M., Verhaeghe, J.C.: Probabilistic behaviour in ants: a strategy of errors? Journal of theoretical Biology **105**(2), 259–271 (1983)
15. Desai, A., Saha, I., Yang, J., Qadeer, S., Seshia, S.A.: DRONA: A Framework for Safe Distributed Mobile Robotics. In: ICCPS (2017). DOI 10.1145/3055004.3055022
16. Di Stefano, L., De Nicola, R., Inverso, O.: Verification of distributed systems via sequential emulation. ACM Transaction on Software Engineering and Methodology **31**(3) (2022). DOI 10.1145/3490387
17. Di Stefano, L., Lang, F.: Verifying temporal properties of stigmergic collective systems using CADP. In: 10th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), *LNCS*, vol. 13036, pp. 473–489. Springer (2021). DOI 10.1007/978-3-030-89159-6_29
18. Di Stefano, L., Lang, F., Serwe, W.: Combining SLiVER with CADP to analyze multi-agent systems. In: 22nd International Conference on Coordination Models and Languages (COORDINATION), *LNCS*, vol. 12134, pp. 370–385. Springer (2020). DOI 10.1007/978-3-030-50029-0_23
19. Dorigo, M., Bonabeau, E., Theraulaz, G.: Ant algorithms and stigmergy. Future generation computer systems **16**(8), 851–871 (2000)
20. Dussutour, A., Fourcassié, V., Helbing, D., Deneubourg, J.L.: Optimal traffic organization in ants under crowded conditions. Nature **428**(6978), 70–73 (2004)
21. Dussutour, A., Nicolis, S.C., Deneubourg, J.L., Fourcassié, V.: Collective decisions in ants when foraging under crowded conditions. Behavioral Ecology and Sociobiology **61**, 17–30 (2006)
22. Emlen, J.T.: Flocking behavior in birds. The Auk **69**(2), 160–170 (1952)
23. Fettke, P., Reisig, W.: Discrete models of continuous behavior of collective adaptive systems. In: 11th International Symposium on Leveraging Applications of Formal Methods (ISoLA), pp. 65–81. Springer (2022)
24. Finkelshtein, D., Kondratiev, Y., Kutoviy, O.: Individual based model with competition in spatial ecology. SIAM Journal on Mathematical Analysis **41**(1), 297–317 (2009). DOI 10.1137/080719376
25. Francalanza, A., Aceto, L., Ingólfsdóttir, A.: On verifying hennessy-milner logic with recursion at runtime. In: 6th International Conference on Runtime Verification (RV), *LNCS*, vol. 9333, pp. 71–86. Springer (2015). DOI 10.1007/978-3-319-23820-3_5
26. Goss, S., Aron, S., Deneubourg, J.L., Pasteels, J.M.: Self-organized shortcuts in the argentine ant. Naturwissenschaften **76**(12), 579–581 (1989)
27. Inverso, O., Trubiani, C.: Parallel and distributed bounded model checking of multi-threaded programs. In: 25th Symposium on Principles and Practice of Parallel Programming (PPoPP), pp. 202–216. ACM (2020). DOI 10.1145/3332466.3374529
28. Koenig, N., Howard, A.: Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), vol. 3, pp. 2149–2154 vol.3. IEEE (2004). DOI 10.1109/IROS.2004.1389727
29. Kouvaros, P., Lomuscio, A.: A counter abstraction technique for the verification of robot swarms. In: 29th Conference on Artificial Intelligence (AAAI), pp. 2081–2088. AAAI (2015)
30. Kuylen, E., Liesenborgs, J., Broeckhove, J., Hens, N.: Using individual-based models to look beyond the horizon: The changing effects of household-based clustering of susceptibility to measles in the next 20 years. In: 20th International Conference on Computational Science (ICCS), *LNCS*, vol. 12137, pp. 385–398. Springer (2020). DOI 10.1007/978-3-030-50371-0_28
31. Lebar Bajec, I., Zimic, N., Mraz, M.: Simulating flocks on the wing: The fuzzy approach. Journal of theoretical biology **233**, 199–220 (2005). DOI 10.1016/j.jtbi.2004.10.003
32. Leucker, M., Schallhart, C.: A brief account of runtime verification. Journal of Logic and Algebraic Programming **78**(5), 293–303 (2009). DOI 10.1016/j.jlap.2008.08.004
33. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: An opensource model checker for the verification of multi-agent systems. Software Tools for Technology Transfer **19**(1), 9–30 (2017). DOI 10.1007/s10009-015-0378-x
34. Mehmood, U., Roy, S., Grosu, R., Smolka, S.A., Stoller, S.D., Tiwari, A.: Neural flocking: MPC-based supervised learning of flocking controllers. In: 23rd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS), *LNCS*, vol. 12077, pp. 1–16. Springer (2020). DOI 10.1007/978-3-030-45231-5_1
35. Monmarché, N., Venturini, G., Slimane, M.: On how pachycondyla apicalis ants suggest a new search algorithm. Future Generation Computer Systems **16**(8), 937–946 (2000). DOI https://doi.org/10.1016/S0167-739X(00)00047-9
36. Olfati-Saber, R.: Flocking for multi-agent dynamic systems: Algorithms and theory. IEEE Transactions on Automatic Control **51**(3), 401–420 (2006). DOI 10.1109/TAC.2005.864190
37. Pinciroli, C., Beltrame, G.: Buzz: An extensible programming language for heterogeneous swarm robotics. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3794–3800. IEEE (2016). DOI 10.1109/IROS.2016.7759558
38. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L.M., Dorigo, M.: ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intelligence **6**(4), 271–295 (2012). DOI 10.1007/S11721-012-0072-5
39. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science

(FOCS), pp. 46–57. IEEE (1977). DOI 10.1109/SFCS.1977.32

40. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.Y.: ROS: An open-source robot operating system. In: ICRA Workshop on Open Source Software (2009)
41. Resnick, M.: Turtles, Termites, and Traffic Jams - Explorations in Massively Parallel Microworlds. MIT Press (1998)
42. Reynolds, C.W.: Flocks, herds and schools: A distributed behavioral model. In: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1987, Anaheim, California, USA, July 27-31, 1987, pp. 25–34. ACM (1987). DOI 10.1145/37401.37406
43. Rohmer, E., Singh, S.P.N., Freese, M.: V-REP: A versatile and scalable robot simulation framework. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1321–1326. IEEE (2013). DOI 10.1109/IROS.2013.6696520
44. Scheibe, V.: Introduction and replication of a bird flocking behavior simulation. Zenodo (2023). DOI 10.5281/ZENODO.8228783. URL `https://zenodo.org/record/8228784`
45. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: 16th International Conference on Computer Aided Verification (CAV), *LNCS*, vol. 3114, pp. 202–215. Springer (2004). DOI 10.1007/978-3-540-27813-9_16
46. Shi, H., Wang, L., Chu, T.: Flocking of multi-agent systems with a dynamic virtual leader. International Journal of Control **82**(1), 43–58 (2009). DOI 10.1080/00207170801983091
47. Stiglitz, J.E., Gallegati, M.: Heterogeneous interacting agent models for understanding monetary economies. Eastern Economic Journal **37**(1), 6–12 (2011). DOI 10.1057/eej.2010.33
48. Sumpter, D.J., Beekman, M.: From nonlinearity to optimality: pheromone trail foraging by ants. Animal behaviour **66**(2), 273–280 (2003)
49. Sumpter, D.J., Blanchard, G.B., Broomhead, D.S.: Ants and agents: A process algebra approach to modelling ant colony behaviour. Bulletin of Mathematical Biology **63**(5), 951–980 (2001). DOI 10.1006/bulm.2001.0252
50. Tofts, C., Hatcher, M., Franks, N.: The autosynchronization of the ant leptothorax acervorum (fabricius): theory, testability and experiment. Journal of theoretical biology **157**(1), 71–82 (1992)
51. Tofts, C.M.N.: Describing social insect behaviour using process algebra. Transaction of the Society for Computer Simulation **9**, 227 (1992)
52. Toner, J., Tu, Y.: Flocks, herds, and schools: A quantitative theory of flocking. Physical Review E **58**(4), 4828–4858 (1998). DOI 10.1103/PhysRevE.58.4828
53. Traniello, J.F.: Foraging strategies of ants. Annual review of entomology **34**(1), 191–210 (1989)
54. Vásárhelyi, G., Virágh, C., Somorjai, G., Tarcai, N., Szörényi, T., Nepusz, T., Vicsek, T.: Outdoor flocking and formation flight with autonomous aerial robots. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3866–3873. IEEE (2014). DOI 10.1109/IROS.2014.6943105
55. Wilensky, U.: Modeling nature's emergent patterns with multi-agent languages. In: EuroLogo (2001)
56. Winfield, A.F.T., Liu, W., Nembrini, J., Martinoli, A.: Modelling a wireless connected swarm of mobile robots. Swarm Intelligence **2**(2-4), 241–266 (2008). DOI 10.1007/s11721-008-0018-0