Intuitive Modelling and Formal Analysis of Collective Behaviour in Foraging Ants*

Rocco De Nicola[1], Luca Di Stefano[2], Omar Inverso[3], and Serenella Valiani[1](✉)

[1] IMT School of Advanced Studies, Lucca, Italy
[2] University of Gothenburg, Gothenburg, Sweden
[3] Gran Sasso Science Institute (GSSI), L'Aquila, Italy
serenella.valiani@imtlucca.it

Abstract. We demonstrate a novel methodology that integrates intuitive modelling, simulation, and formal verification of collective behaviour in biological systems. To that end, we consider the case of a colony of foraging ants, where, for the combined effect of known biological mechanisms such as stigmergic interaction, pheromone release, and path integration, the ants will progressively work out the shortest path to move back and forth between their nest and a hypothetical food repository. Starting from an informal description in natural language, we show how to devise intuitive specifications for such scenario in a formal language. We then make use of a prototype software tool to formally assess whether such specifications would indeed replicate the expected collective behaviour of the colony as a whole.

Keywords: Agent-based models · Collective behaviour · Foraging · Ant colonies · Simulation · Formal verification.

1 Introduction

Researchers in systems biology, ecology, and countless other disciplines have long since been interested in computational methods to rapidly explore promising ideas and hypotheses without having to resort to controlled real-world experiments or field studies. More specifically, agent- or individual- based models have been suggested as an effective research aid across several research areas [5,19]. In these models, the system is represented as a collection of autonomous, interacting components (or *agents*), whose interaction both lead to, and may be influenced by, the *emergence* of collective phenomena.

This form of modelling is becoming increasingly popular, owing to the continuous growth in available computational power and its ability to faithfully reproduce non-linear and complex dynamics [27,33]. A common approach to so-called agent-based modelling [1,12,23,39] is to describe the individual agents

* 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*.

through mathematical formalisms, e.g., interconnected continuous- or discrete-time dynamical systems. This does have its benefits, as it allows leveraging powerful analytical tools developed in well-established areas of mathematical research, and in particular allow simulation through efficient numerical methods. However, it requires a considerable learning curve, and is hardly suited to incremental or compositional modelling.

We argue that adopting a tailored formal language may provide better affordances, enabling an intuitive modelling methodology where informal ideas and concepts related to the behaviour of a system gradually evolve into machine-tractable models expressed through formal specifications. Such models can be maintained and refined with limited effort, for instance to introduce small variations in the attempt to prove or disprove specific hypotheses on a system of interest. Besides, adopting a language with formal semantics opens up to a variety of possibilities in terms of formal analysis techniques, and can thus offer considerably more rigorous forms of reasoning than possible with massive simulation [14].

Our methodology (sketched in Fig. 1) aims at supporting intuitive agent-based modelling of the system of interest, while reducing the overall technical effort for automated analysis. In practice, one starts by identifying the relevant characteristics of the environment and of the agents populating it. Then, one sketches an informal description of the behaviour of the agents in natural language. Such a description identifies the main actions carried out by the agents, and the effect of such actions on their own characteristics and on the environment. Then, taking advantage of a domain-specific language, one progressively expands such main actions into more detailed sequences of operations, eventually working out full formal specifications. Thanks to a mechanised procedure, at this point the formal specifications can either be simulated or formally verified. Simulation is less computationally expensive than verification, and allows for quick feedback initially. By inspecting simulated system evolutions, one can quickly identify and fix possible problems, such as situations that are not realistic, or trivial violations of some simple property of interest. After one or more rounds of refinements, one can start to analyse the emerging behaviour of interest, again via more extensive simulations, or formal verification, and so on, possibly until the verification verdict is satisfactory.

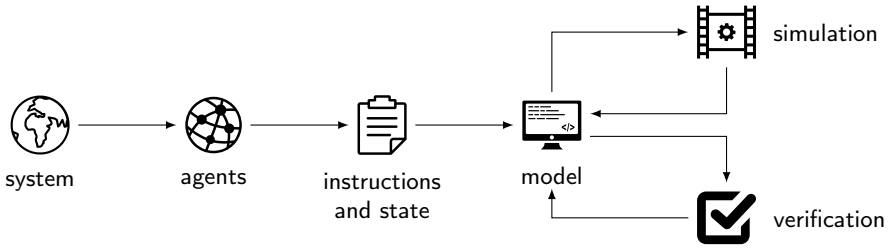

Fig. 1: A visualization of our proposed methodology.

To demonstrate our methodology, we focus on the scenario of foraging ants, where multiple known biological mechanisms, such as *stigmergic interaction* [34,28], *pheromone release* [16], and *path integration* [38], take place. Starting from an informal description of the behaviour of the ants in plain English, we gradually obtain behavioural specifications in an existing formal language [13]. We then wonder whether our ant colony model would somehow be able replicate the well-known *emerging behaviour* observed by Goss et al. [22], whereby the ants will gradually work out the shortest path to move back and forth between the nest and the food repository. To that end, we discuss how state-of-the-art general-purpose techniques for automated analysis [17] can fit within our methodology and be automatically re-used to either *simulate* or *formally verify* the specifications. The possibility of integrating simulation and formal analysis within the same workflow is a particularly relevant element of novelty in this context: on the one hand simulation allows to quickly evaluate possible hypotheses of interest on the system under analysis; on the other hand, formal analysis provides a considerably more rigorous formalism that can effectively pinpoint so-called *rare events* which usually are difficult to pinpoint, or prove their absence for good.

The paper is structured as follows. In Section 2 we introduce the scenario of a colony of foraging ants and develop our formal specifications for it. In Section 3 we show how to analyse the specifications in order to assess the emergence of the mentioned collective behaviour of interest in the colony. We discuss related work in Section 4 and provide our concluding remarks in Section 5.

2 Modelling

In this section, we develop a model to replicate the collective behaviour of a colony of foraging ants, where different mechanisms extensively studied in biology take place. Starting from an informal description of the behaviour of interest in English, we gradually obtain intuitive specifications in a formal language known as LAbS [13].

Let us consider a colony of ants initially moving chaotically while searching for food. Over time, the ants are progressively influenced by the presence of pheromone in the environment: an ant may smell traces of pheromone nearby, and move towards it, following the mechanism of *stigmergic interaction* observed in several species [34,28]. Having picked up food, the ant will return to its nest using the most efficient route thanks to a well-known ability known as *path integration* [38]; at the same time, the ant releases pheromone on its way back to the nest, following the pattern of *pheromone release* observed in [16].

Let us now focus on the *environment* in which the colony operates. We assume that the space is a two-dimensional *arena* of regular shape, initially without traces of pheromone; we also assume for simplicity that there exists a single source of food, and that there are no obstacles. In Listing 1, we model the arena as a square grid of size *size* and the coordinates of the food repository as $(foodx, foody)$. We represent the amount of pheromone at each position of the arena as an array

Listing 1: Definition of the environment and external parameters.

```
1 system {
2   extern = size, n, foodx, foody, m, k
3   environment = ph[size, size]: 0
4   spawn = Ant: n
5 }
```

Listing 2: Attributes of an ant.

```
1 agent Ant {
2   interface = x: 0..size; y: 0..size;
3               nextX: 0; nextY: 0
4   ...
5 }
```

initially containing only zeros (line 3). We eventually specify that our colony consists of n ants (line 4). Note that $foodx, foody, size$, and n are parameters of the specifications whose value is to be set by the user at the beginning of the analysis; the same holds for m (that constrains the range of movement for every ant), and for k (discussed later).

To model the position and movement of an ant, we introduce separate features (or *attributes*) to keep track of its current position and the position it wishes to move to. We assume that initially every ant starts from any position within the arena. Intuitively, here we are assuming that the ants have already been exploring the arena for a while, and thus might have ended up anywhere in the arena. Listing 2 shows how this is formalized in LAbS. At line 2, x and y indicate the coordinates of the ant within the grid; the notation $0..size$ indicates a non-deterministic value in the interval ranging from 0, inclusive, to $size$, exclusive. Lines 3 initialize the coordinates of the position where the ant wishes to move.

We can now describe the behaviour of an ant. Following the informal description at the beginning of the section, an ant repeatedly explores the arena until it finds some food, heads back to its nest, and finally starts over. This is formalized in Listing 3, where an ant's behaviour (Behavior) is described as a sequence of steps, or, more formally, *processes*: one for exploring (Explore), one for returning to the nest (GoHome), and one (Behavior) which is actually a recursive call to repeat the same sequence indefinitely. The rest of the section is dedicated to formalizing such three processes; from now on our listings will omit the other parts of the ants' specification and only show the behavioural definitions.

The Explore process is composed of a sensing phase during which the ant looks for nearby pheromone, and a moving phase when the ant either goes to a pheromone-marked position or, if none is sensed, moves chaotically. When following the GoHome behaviour, the ant marks its current position with pheromone and then moves one step closer to the nest (as mentioned earlier, we can assume that an ant knows the right way home thanks to path integration).

Listing 4 expands the specifications of the process Explore and GoHome. Lines 6–10 shows the details for Explore. The exploration consists in first searching for pheromone and then moving, but only as long as the ant is not at the same position as the food source. We enforce this requirement at line 7: if it does not hold, the rest of the process is skipped. The definition of SmellPheromone in Listing 4 involves two additional processes, namely SmellPheromone and Move

Listing 3: High-level behaviour of an ant.

```
1 agent Ant {
2   interface = ...
3
4   Behavior =
5     Explore;
6     GoHome;
7     Behavior
8 }
```

(lines 8–9), followed by a self-loop, to make sure the process loops until the ant finds itself at the food repository, at which point it will break out of the process.

Let us now expand on SmellPheromone. Initially, an ant searches for nearby traces of pheromone considering some position at maximum sensing distance m and then checking whether it contains a pheromone marking. Listing 4 (lines 24–40) encodes such behaviour. Observe that the process description is enclosed in curly braces: this specifies that all actions therein are executed by the ant in a single step. Lines 25–26 pick two a non-deterministic value between 1 and $m + 1$, which are used to select two positions within the sensing distance of the ant (textx1, texty1) and (textx2, texty2) (lines 27–30). To ensure that they are within the grid, their values are compared and only those between 0 and $size - 1$ are accepted. Lines 32–39 define the position (nextX, nextY) that the ant will move to if it detects pheromone. Note that, if the ant does not detect pheromone at either of the two observed positions, the desired position is the current position of the ant.

Listing 4 (lines 12–21) defines the actual movement of the ant. If the ant has detected pheromone at one of the two previous positions, the content of the block is ignored and the position of the ant updated with the desired position (line 22). In case no pheromone was detected, the ant chooses a reachable position nearby (lines 14–16) within the grid (lines 17–20) and updates its position with the chosen one (line 22). It is worth to note that the way we implement the pheromone detection allows for some uncertainty: even when very close to a pheromone-marked location, an ant may still fail to detect it and stray from the trail. Once the ant reaches the food, it starts heading back to the nest. For simplicity, we position the nest at one of the edges of the arena, at $(0, foody)$, so that the shortest path between the nest and food is a straight horizontal segment. Lines 42–46 implement the homing behaviour: the ant releases a quantity of pheromone equal to 1 (line 44) and moves towards the nest (line 45), and repeats these two steps until the nest is reached.

3 Analysis

We would now like to assess whether, following the behaviour formalised in Sect. 2, our ants will keep wandering chaotically through the arena, or instead

Listing 4: Full behavioural description of an ant.

```
Behavior =
  Explore;
  GoHome;
  Behavior

Explore =
  x ≠ foodx or y ≠ foody ⇒ (
    SmellPheromone;
    Move;
    Explore)

Move =
  (nextX = x and nextY = y ⇒ {
    dX, dY := [−m..m+1], [−m..m+1];
    nextX ← x+dX;
    nextY ← y+dY;
    nextX ← max(nextX, 0);
    nextY ← max(nextY, 0);
    nextX ← min(nextX, size−1);
    nextY ← min(nextY, size−1)
  });
  x, y ← nextX, nextY

SmellPheromone = {
  dX := [1..m+1];
  dY := [1..m+1];
  testx1, testy1 := min(x+dX, size−1),
                    min(y+dY, size−1);
  testx2, testy2 := max(x−dX, 0),
                    max(y−dY, 0);

  nextX ←
    if ph[testx1, testy1] then testx1 else
    if ph[testx2, testy2] then testx2
    else x;
  nextY ←
    if ph[testx1, testy1] then testy1 else
    if ph[testx2, testy2] then testy2
    else y
}

GoHome =
  x ≠ 0 or y ≠ foody ⇒ ({
  ph[x,y] ⇐1;
  x ← max(0, x−1)
  }; GoHome)
```

will gradually prefer the shortest path between their nest and the food source, thus replicating an emerging phenomenon similar to that observed by Goss et al. [22].

To enable automated reasoning, we decorate the specifications with appropriate properties that allow to evaluate whether such behaviour of interest emerges. Knowing that the shortest path is the segment from $(0, \mathit{foodx})$ to $(\mathit{foodx}, \mathit{foody})$, we can look for evidence of emergence by evaluating how far our ants are from this segment as the system evolves. If the system is able to self-organize, this distance should gradually decrease with time; otherwise, it should fluctuate chaotically. For an ant located at position (x, y), we define its distance from the path as:

$$d(\mathsf{x}, \mathsf{y}) = |\mathsf{y} - \mathit{foody}| + \max(0, \mathsf{x} - \mathit{foodx}),$$

meaning that the distance is just $|\mathsf{y} - \mathit{foody}|$ if the ant's x-coordinate is not greater than foodx; otherwise, it is the Manhattan distance [7] between the ant and the food source.

We can now proceed to analysing the colony of ants in different ways. We first generate arbitrary simulation traces, to gather some *empirical* evidence about our

Listing 5: Assumptions on the initial state of the ants.

```
1 assume {
2   FoodAnt = exists Ant a,
3     (x of a = foodx) and (y of a = foody)
4   FarFromThePath = forall Ant a,
5     ((x of a = foodx) and (y of a = foody)) or
6     (x of a > foodx + k) or
7     (y of a > foody + k) or
8     (y of a < foody − k)
9 }
```

Listing 6: Checking that ants aggregate close to the shortest path after B steps have elapsed.

```
1 check {
2   ShortestPath =
3     after B forall Ant a,
4     (x of a ≤ foodx + k) and
5     (y of a ≥ foody − k) and
6     (y of a ≤ foody + k)
7 }
```

hypothesis; then, we use formal verification to understand whether the collective behaviour in consideration is *guaranteed* to happen.

Let us define some concepts used in the rest of the section. A *trace* of a system is a sequence of *steps* performed by its agents. A step corresponds to the execution of a single statement, or a compound atomic block enclosed in braces. We focus on finite-length traces, where ants perform their steps one at a time in a fixed order. Thus, a trace of length $q \cdot n$ for a system composed of n ants will consist of q steps per ant, and we say that this execution consists of q *epochs*.

Verification technology. To perform our experiments, we use a tool called SLiVER[4] that enables *simulation* as well as *formal verification* of LAbS models using a variety of techniques. The core idea behind SLiVER is to turn the LAbS specification given as input into a symbolic intermediate representation that can be translated into different target languages, so as to re-use off-the-shelf tools developed for those languages as back ends for the analysis [17]. For the scope of this work, we rely on symbolic bounded model checking of [10], which in turn reduces to Boolean satisfiability (SAT). We would like to emphasize that we use the same encoding and rely on the same back end technique for both simulation and verification: for the former, we override some of the heuristics of the SAT solver in order to introduce extra randomness and thus produce different simulations at each run.

Assumptions on the initial state. In our specifications, ants are initially scattered through the arena (see line 2 of Listing 2). At the same time, we want to start our simulation at a moment where *at least* one ant has found the food source: after all, we already know that until some pheromone appears in the arena, ants can only carry on with their chaotic exploration. Finally, we want to ignore those simulations where the ants that have not found food are too close to the shortest path between the food repository and the nest. This last requirement is simply there so that we can focus on more interesting traces.

[4] SLiVER is available at `https://github.com/labs-lang/sliver`.

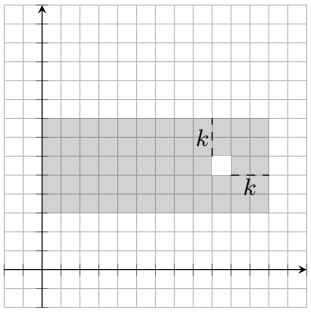

Fig. 2: The FarFromThePath assumption of Listing 5 enforces that no ant will occupy the shaded area in the initial state. The unshaded square within the shaded area is the location of the food source.

We formalize these requirements in a separate section of the specifications, as shown in Listing 5. This separation enables us to quickly evaluate different scenarios without having to alter the behaviour of the ants. Here, we say that at least one ant starts at the same position of the food source (AntOnFood), and we define a region around the shortest food-nest path that ants cannot occupy in the initial state (FarFromThePath). This region is depicted in Figure 2.

Simulation. Given a file `ants.labs` containing our specifications, we can ask SLiVER to generate s traces of length B by invoking it as follows:

```
./sliver.py ants.labs <parameters> --fair --simulate s --steps B
                    --concretization=sat
```

Here, `<parameters>` is a list of values to assign to each external parameter, and `--fair` enforces that agents execute their actions in ordered epochs. The `--concretization=sat` option enables a *concretization* step that alleviates the load on the SAT solver underlying our workflow. This step determines a feasible, random initial state of the system, and instruments the solver to start from that specific state, rather than considering all feasible initial states indifferently. The same instrumentation also involves other parts of the model, e.g., the operations at lines 25–26 and 14 of Listing 4, which pick values nondeterministically.[5]

We ran 200 simulations 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. Table 1 shows the parameters used to instantiate the experiments. Each generated trace had a length of 800 steps, i.e., 80 epochs. By running 16 instances of SLiVER in parallel, we were able to simulate our system at an approximate rate of one trace

[5] Notice that this is not the same as solving the formula under assumptions: if the instrumentation makes the formula unsatisfiable, we leave the solver free to drop part of it and resume its search.

Table 1: Values for our model's parameters during the experimental evaluation.

Name	**Description**	**Value**
foodx, *foody*	Position of the food source	(10, 10)
k	Initial distance from the shortest path	2
n	Number of ants	10
m	Maximum range of an ant's movement	1

every 13 minutes. This will seem quite slow for simulation. However, we would like to remark that the focus here is on integrating simulation and formal analysis within the same workflow; our workflow was initially conceived for exhaustive formal analysis and is not optimised for simulation yet.

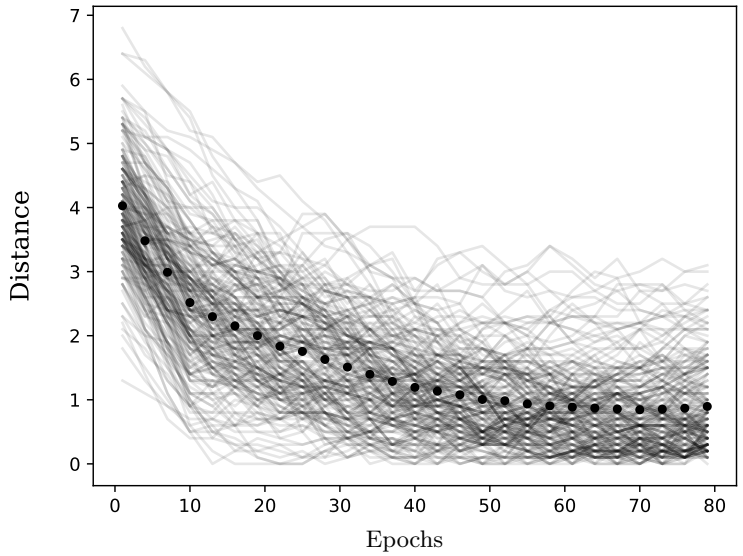

Fig. 3: Average distance of the ants from the shortest nest-food path.

The results of our simulations are summarized in Fig. 3. Each line depicts the average ant-path distance for one trace. Circle markers denote the mean value across all traces. The plot does seem to indicate that the average distance between the ants and the shortest path decreases with time. Apart from some outliers, this distance appears to drop below 2 rather quickly: the mean across all traces becomes less than 2 after 21 epochs, and becomes 0.6 after 80 epochs. The initial average distance may vary across traces because we allow one, two or

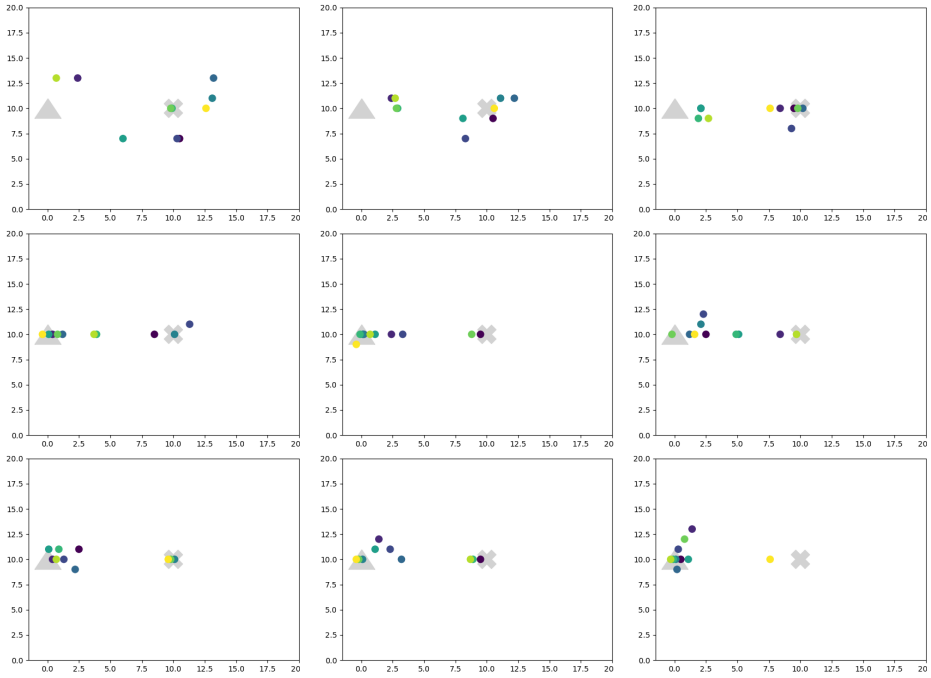

Fig. 4: A counterexample to the property of Listing 6.

more ants to start at the food source location: thus, their initial distance from the shortest path is nought.

Formal verification. We now start to wonder whether the behaviour observed in the simulations is *inevitably* going to happen. In other words, will *every* trace of this system show the emergence of this feature?

To answer this question, we decorate our specifications with an additional section, containing the property that we wish to verify. In this case, we want all ants to be close enough to the shortest path after they have performed a certain number of actions B, which in our case corresponds to the verification bound. By "close enough" we mean that their value of `x` should be at most $foodx + k$ and that their distance from the path should not exceed k: (Listing 6). In other words, they should all lie within the area they were forbidden to occupy in the initial state, which we depicted in Figure 2.

To verify this property, we invoke SLiVER with the following command line:

`./sliver.py ants.labs <parameters> --fair --steps` B.

We ran this task using the parameters in Table 1 and the same bound of $B = 800$ steps, i.e., 80 epochs. Our verification task fails and SLiVER produces a counterexample as proof that the system can indeed violate the property under

Listing 7: Ensuring that a single ant is initially located at the food source.

```
assume {
  UniqueFoodAnt = exists−unique Ant a,
    onFood of a = 1

  OneOnFood = forall Ant a,
    (onFood of a = 0) or
    ((x of a = foodx) and (y of a = foody))

  OthersFarFromThePath = forall Ant a,
    (onFood of a = 1) or
    (x of a > foodx + k) or
    (y of a > foody + k) or
    (y of a < foody − k)
}
```

Listing 8: The negation of the property shown in Listing 6.

```
check {
  NegShortestPath = after B
    exists Ant a,
    (x of a > foodx + k) or
    (y of a > foody + k) or
    (y of a < foody − k)
}
```

investigation, in other words, the ants will not necessarily adopt an increasingly optimised food-nest path.

Figure 4 shows a graphical visualization of the counterexample (we only report one state every 10 epochs). Each circle represents an ant; the triangle and cross mark the position of the nest and food source, respectively. As the trace shows, the ants still appear to aggregate around the shortest path, but at some point one of them strays away and ends up in position $(1, 13)$, i.e., at a distance of 3 from the path. This is enough to violate the property. This counterexample is a result of the uncertainty in the ants' pheromone detecting behaviour. Probabilistically, it is rather likely that an ant close to the pheromone trail will detect it at least once in a while, and thus keep close to it: this explains why our previous simulations seem to converge rather effortlessly. However, formal verification treats all potential traces alike, regardless from their likelihood in a probabilistic sense. This allows SLiVER to detect traces that could be hard to spot through simulation alone.

Looking for interesting traces via verification. We close this section by highlighting another potential use of verification: that is, to assess the *existence* of a trace satisfying some specific requirements. For instance, let us consider again our specifications and ask ourselves: assuming that *a single* ant starts at the food source, is there *any* trace where all ants manage to aggregate along the shortest path? We can get an answer by asking SLiVER to verify that *all* traces satisfy the *negation* of this property: if this task fails, we will be presented with a trace that satisfies our requirements. Otherwise, it means that no such trace exists.

To encode the new assumptions, we equip all ants with an attribute onFood that may be initialized to either 0 or 1. Then, we replace our previous assumptions with new ones (Listing 7) stating the following: only one ant starts with onFood set

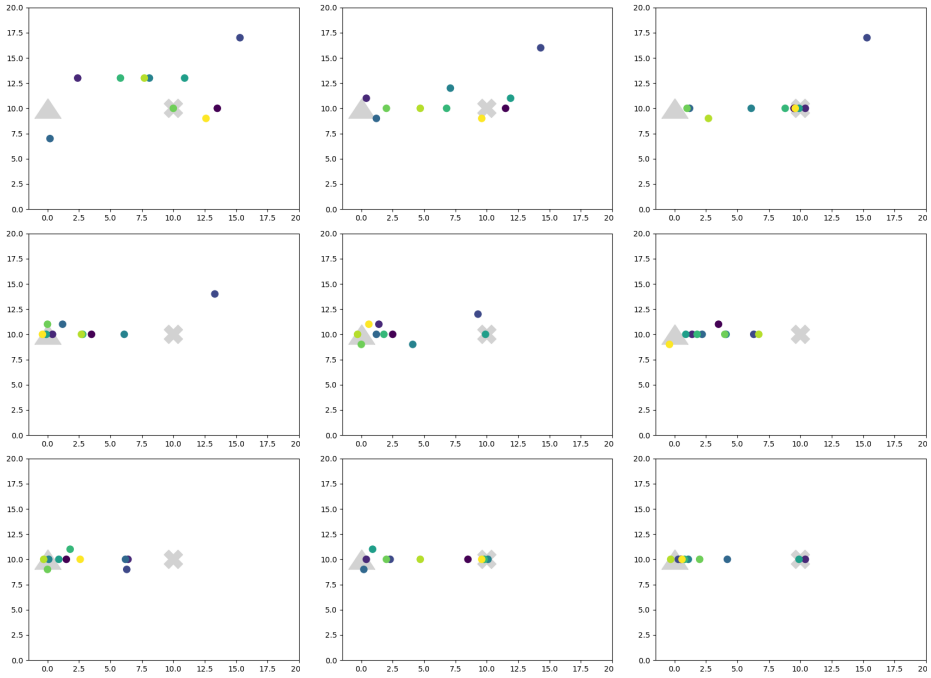

Fig. 5: A counterexample to the property of Listing 8.

to 1 (UniqueFoodAnt); this ant starts at the food location (OneOnFood); and all other ants start somewhere far from the shortest path (OthersFarFromThePath).[6] Lastly, we have to negate the property that we have previously verified: that is, we want to check that after B steps, there is at least one ant that is still far from the shortest path. The resulting property is shown in Listing 8.

We verified this property on the same machine used for the previous verification task, with the same parameters and verification bound. Again, the task failed and SLiVER returned the counterexample depicted in Figure 5.

4 Related Work

Extensive overviews of the foraging strategies followed by different species of ants can be found in [16,52]. For additional discussion about current research on ant foraging, we refer the reader to [28].

Foraging, as well as other elements of ant colony behaviour, is amenable to *macroscopic* modelling, which describes a system in terms of its aggregate features while losing most information about the individuals that compose it. The seminal work by Goss *et al.* on self-organized shortcuts, for instance, relies on a random

[6] Adding the attribute onFood is not strictly necessary, but makes for more readable specifications.

process model [22]. Other works describe the colony through reaction-diffusion equations [48], or as a dynamical system with excitability dynamics [40]. These models appear to focus on one single aspect of the colony's collective behaviour. This allows their authors to choose the mathematical objects and structures that provide the best fit for the modelled scenario; however, this specialization also means that different models of the same system may rely on entirely different, possibly hard to reconcile approaches. As a consequence, composing two or more models of this kind (say, to study whether the modelled aspects may interfere with each other) is likely to require extensive effort.

Agent-based mathematical models of foraging are also abundant in the literature [34,43,53]. Additionally, the entire field of *ant colony optimization* [18] originates as an application of an agent-based colony model to optimization problems, where artificial ants explore an arena that encodes the problem space and use pheromone markings to identify the optimal solution.

Our work has deep ties to *process algebras*. These formalisms were originally introduced as a tool to reason about concurrency-related problems in computer science [3,24,37], but have long since been suggested as an effective tool for modelling complex natural systems. Among other things, it is argued that process algebras are inherently compositional (so they lend themselves well to describing collections of interacting agents), and that the specifications can be formally validated [51].

Our formal specification language LAbS aims at exploiting these perks, while offering a more high-level and intuitive set of primitives and constructs for the specification of collective systems. Other forms of natural behaviour that have been modelled with LAbS include bird flocking [14] and a simplified reproduction of the experiment from [22] on self-organization in colonies of the Argentinian ant *Iridomyrmex humilis* [15]. However, these models were somewhat limited since they used an earlier version of the language without conditional processes and multidimensional arrays.

The semantics of a process algebra is commonly defined as some form of transition system (LAbS also falls in this category). This allows for verification of qualitative properties by exhaustively exploring the state space of the system. It is also possible to equip a process algebra with probabilistic or quantitative semantics, thus enabling other forms of automated reasoning: relevant examples include WSCCS [49], which has a Markov-chain semantics and has been used to model and analyse behaviours commonly observed in ant colonies, from synchronization to activity recruiting [47,50] and Bio-PEPA [9], with an ODE-based semantics that has been exploited for fluid flow analysis of ant foraging behaviour [36]. For an exhaustive list of Process Algebra-based methods, please refer to Section 2 of [2].

The programming and modelling language literature provides a wide choice of languages tailored towards collective, agent-based systems [29]. Examples include ASCAPE [26], SCAMP [42], and MASON [35]; see [41] for a MASON model of foraging ants that bears some similarity to our work.

Arguably, the most notable specimens in this category are LOGO [20] and its derivatives, most notably StarLogo [44] and NetLogo [54], which have found a rather wide adoption as a tool to create and simulate agent-based models across multiple research areas, including systems biology [8,43]. StarLogo and NetLogo are engineered to handle thousands of agents, and NetLogo also integrates powerful visualization features. Additionally, StarLogo and NetLogo allow specifying complex, dynamic environments by defining *patches*, a concept introduced in the artificial life community [25]. The environment is assumed to be an n-dimensional grid partitioned into non-overlapping patches or cells, where each patch may interact with neighbouring patches and with agents that are inhabiting it at a given moment. Thus, the concept of patches reconciles agent-based models with cellular automata [31]. In contrast, the world inhabited by LAbS agents has no predefined structure and is essentially just a store of shared variables (e.g., the pheromone matrix in our case study): the structure, if any, emerges from the behavioural rules of the individual agents.

5 Conclusions

In this work we have advocated for the application of language-based, bottom-up methodologies to support research in computational biology, and highlighted the benefits of integrating simulation and formal analysis within the same workflow. We have shown that high-level languages with adequate constructs and primitives allow for an intuitive process where an informal description of the individual behaviour of a complex biological system (in our example, an ant colony) is gradually turned into a formal specification. Compared to lower-level mathematical models, these specifications are easier to refine and extend; they support reuse and composition; furthermore, they are amenable to formal verification, which may provide further insight into the system's emergent properties.

To illustrate this, we have first produced a collection of simulation traces apparently confirming the emergence of self-organized pathfinding in the colony. Then, we have shown how formal verification can disprove that this form of behaviour emerges inevitably. Intuitively, this is due to the fact that ants in our model are able to take a sequence of bad choices that lead them farther, not closer, to the path. Lastly, we have used verification to show that a single ant finding a food source may be enough for the self-organizing behaviour to emerge. Crucially, these tasks were all carried out starting from the same model, with only minimal and intuitive changes being required for the last task.

We plan on improving our methodology along several directions. The modelling language should be further extended with primitives promoting composability and code reuse, such as parameterized process invocations and modules. We would like to foster cooperation with the computational biology community, to gather realistic case studies and to get valuable feedback to motivate further refinements of our language and methodologies.

On the analysis side, we intend to address the main drawback of our simulation workflow, namely its limited efficiency with respect to traditional approaches

based either on executing the model (when the development platform includes a compiler or runtime environment) or on numerical methods. In this respect, SLiVER's reliance on existing verification technologies allows it to evolve with little effort as the state of the art in general-purpose automated reasoning advances. For instance, we could improve the performance by integrating recent optimised SAT decision procedures [21], or by exploiting techniques for deep counterexample detection, e.g., transition power abstractions [4]. This might be especially beneficial when simulating long traces.

Probabilistic reasoning may be enabled by leveraging existing probabilistic model checkers [30]. To support this use case, we might extend the language so that the choices of agents follow a probabilistic distribution. Lightweight simulation-based formal techniques, including statistical model checking [45] and runtime verification [32], may provide some statistical assurance on the behaviour of systems that are currently out of the reach of formal analysis.

In this work, we have focused on bounded executions of systems with a fixed and predefined number of agents. We may support unbounded verification either by computing a *completeness threshold*, i.e., a bound large enough to encompass the whole state space of the system under analysis [11], or by resorting to inductive reasoning techniques for program analysis, such as k-induction [46] or property-directed reachability (PDR, also known as IC3) [6]. We could move past systems of predefined size by considering the number of agents as a fixed, but of arbitrary value. In principle, we could even add language primitives allowing an agent to *spawn* others, or to *leave* the system, making the size of our models dynamic. At the analysis level, these changes would likely require describing our agents through dynamically-allocated data structures; while this is feasible in principle, we suspect that it would result in further challenges to tractability.

References

1. Attanasi, A., Cavagna, A., Del Castello, L., Giardina, I., Grigera, T.S., Jelić, A., Melillo, S., Parisi, L., Pohl, O., Shen, E., Viale, M.: Information transfer and behavioural inertia in starling flocks. Nature Physics **10** (2014). https://doi.org/10.1038/nphys3035
2. Bartocci, E., Lió, P.: Computational modeling, formal analysis, and tools for systems biology. PLoS computational biology **12**(1), e1004591 (2016)
3. Bergstra, J.A., Klop, J.W., Tucker, J.V.: Algebraic tools for system construction. In: Workshop on Logics of Programs. LNCS, vol. 164, pp. 34–44. Springer (1983). https://doi.org/10.1007/3-540-12896-4_353
4. Blicha, M., Fedyukovich, G., Hyvärinen, A.E.J., Sharygina, N.: Transition power abstractions for deep counterexample detection. In: Fisman, D., Rosu, G. (eds.) TACAS. LNCS, vol. 13243, pp. 524–542. Springer (2022). https://doi.org/10.1007/978-3-030-99524-9_29
5. Bonabeau, E.: Agent-based modeling: Methods and techniques for simulating human systems. PNAS **99** (2002). https://doi.org/10.1073/pnas.082080899
6. Bradley, A.R.: SAT-based model checking without unrolling. In: VMCAI. LNCS, vol. 6538, pp. 70–87. Springer (2011). https://doi.org/10.1007/978-3-642-18275-4_7

7. Brezis, H., Brézis, H.: Functional analysis, Sobolev spaces and partial differential equations, vol. 2. Springer (2011)
8. Chiacchio, F., Pennisi, M., Russo, G., Motta, S., Pappalardo, F.: Agent-based modeling of the immune system: NetLogo, a promising framework. BioMed Research International **2014** (2014). https://doi.org/10.1155/2014/907171
9. Ciocchetta, F., Hillston, J.: Bio-PEPA: An extension of the process algebra PEPA for biochemical networks. Electr. Notes Theor. Comput. Sci. **194** (2008). https://doi.org/10.1016/j.entcs.2007.12.008
10. Clarke, E., Kroening, D., Lerda, F.: A tool for checking ANSI-C programs. In: TACAS. pp. 168–176. LNCS, Springer (2004). https://doi.org/10.1007/978-3-540-24730-2_15
11. Clarke, E.M., Kroening, D., Ouaknine, J., Strichman, O.: Completeness and Complexity of Bounded Model Checking. In: VMCAI. LNCS, vol. 2937, pp. 85–96. Springer (2004). https://doi.org/10.1007/978-3-540-24622-0_9
12. Cristiani, E., Menci, M., Papi, M., Brafman, L.: An all-leader agent-based model for turning and flocking birds. J. Math. Biol. **83** (2021). https://doi.org/10.1007/s00285-021-01675-2
13. De Nicola, R., Di Stefano, L., Inverso, O.: Multi-agent systems with virtual stigmergy. Sci. Comput. Program. **187** (2020). https://doi.org/10.1016/j.scico.2019.102345
14. De Nicola, R., Di Stefano, L., Inverso, O., Valiani, S.: Modelling flocks of birds from the bottom up. In: ISoLA. LNCS, vol. 13703, pp. 82–96. Springer (2022). https://doi.org/10.1007/978-3-031-19759-8_6
15. De Nicola, R., Di Stefano, L., Inverso, O., Valiani, S.: Modelling flocks of birds and colonies of ants from the bottom up. Int. J. Softw. Tools Technol. Transf. (2023, to appear)
16. Deneubourg, J.L., Aron, S., Goss, S., Pasteels, J.M., Duerinck, G.: Random behaviour, amplification processes and number of participants: How they contribute to the foraging properties of ants. Physica D **22**(1), 176–186 (1986). https://doi.org/10.1016/0167-2789(86)90239-3
17. Di Stefano, L., De Nicola, R., Inverso, O.: Verification of distributed systems via sequential emulation. ACM Trans. Softw. Eng. Methodol. **31** (2022). https://doi.org/10.1145/3490387
18. Dorigo, M., Birattari, M., Stützle, T.: Ant colony optimization. IEEE Comput. Intell. Mag. **1** (2006)
19. Farmer, J.D., Foley, D.: The economy needs agent-based modelling. Nature **460** (2009). https://doi.org/10.1038/460685a
20. Feurzeig, W., Papert, S.: Programming-languages as a conceptual framework for teaching mathematics. In: NATO Conference on Computers and Learning. pp. 37–42 (1968)
21. Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M.: SAT competition 2020. Artif. Intell. **301**, 103572 (2021). https://doi.org/10.1016/j.artint.2021.103572
22. Goss, S., Aron, S., Deneubourg, J.L., Pasteels, J.M.: Self-organized shortcuts in the Argentine ant. Naturwissenschaften **76**(12), 579–581 (1989). https://doi.org/10.1007/BF00462870
23. Grauwin, S., Bertin, E., Lemoy, R., Jensen, P.: Competition between collective and individual dynamics. PNAS **106** (2009). https://doi.org/10.1073/pnas.0906263106
24. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
25. Hogeweg, P.: Mirror beyond mirror: Puddles of life. In: ALIFE. Santa Fe Institute Studies in the Sciences of Complexity, vol. 6, pp. 297–316. Addison-Wesley (1987)

26. Inchiosa, M.E., Parker, M.T.: Overcoming design and development challenges in agent-based modeling using ASCAPE. PNAS **99** (2002). https://doi.org/10.1073/pnas.082081199
27. Kaul, H., Ventikos, Y.: Investigating biocomplexity through the agent-based paradigm. Briefings in Bioinformatics **16** (2015). https://doi.org/10.1093/bib/bbt077
28. Kolay, S., Boulay, R., d'Ettorre, P.: Regulation of ant foraging: A review of the role of information use and personality. Frontiers in Psychology **11**, 734 (2020). https://doi.org/10.3389/fpsyg.2020.00734
29. Kravari, K., Bassiliades, N.: A survey of agent platforms. Journal of Artificial Societies and Social Simulation **18** (2015). https://doi.org/10.18564/jasss.2661
30. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. LNCS, vol. 6806, pp. 585–591. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_47
31. Langton, C.G.: Studying artificial life with cellular automata. Physica D **22**(1), 120–149 (1986). https://doi.org/10.1016/0167-2789(86)90237-X
32. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebr. Program. **78** (2009). https://doi.org/10.1016/j.jlap.2008.08.004
33. Levin, S.: Complex adaptive systems: Exploring the known, the unknown and the unknowable. Bull. Amer. Math. Soc. **40** (2003). https://doi.org/10.1090/S0273-0979-02-00965-5
34. Li, L., Peng, H., Kurths, J., Yang, Y., Schellnhuber, H.J.: Chaos–order transition in foraging behavior of ants. PNAS **111** (2014). https://doi.org/10.1073/pnas.1407083111
35. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.C.: MASON: A multiagent simulation environment. Simulation **81** (2005). https://doi.org/10.1177/0037549705058073
36. Massink, M., Latella, D.: Fluid analysis of foraging ants. In: COORDINATION. LNCS, vol. 7274, pp. 152–165. Springer (2012). https://doi.org/10.1007/978-3-642-30829-1_11
37. Milner, R.: A Calculus of Communicating Systems, LNCS, vol. 92. Springer (1980). https://doi.org/10.1007/3-540-10235-3
38. Müller, M., Wehner, R.: Path integration in desert ants, *cataglyphis fortis*. PNAS **85**(14), 5287–5290 (1988). https://doi.org/10.1073/pnas.85.14.5287
39. Olfati-Saber, R.: Flocking for multi-agent dynamic systems: Algorithms and theory. IEEE Trans. Automat. Contr. **51** (2006). https://doi.org/10.1109/TAC.2005.864190
40. Pagliara, R., Gordon, D.M., Leonard, N.E.: Regulation of harvester ant foraging as a closed-loop excitable system. PLoS Comput. Biol. **14**(12) (2018). https://doi.org/10.1371/journal.pcbi.1006200
41. Panait, L.A., Luke, S.: Ant foraging revisited. In: ALIFE. pp. 569–574. MIT Press (2004). https://doi.org/10.7551/mitpress/1429.003.0096
42. Parunak, H.V.D.: Social simulation for non-hackers. In: MABS Revised Selected Papers. LNCS, vol. 13128, pp. 1–14. Springer (2021). https://doi.org/10.1007/978-3-030-94548-0_1
43. Perna, A., Granovskiy, B., Garnier, S., Nicolis, S.C., Labédan, M., Theraulaz, G., Fourcassié, V., Sumpter, D.J.T.: Individual rules for trail pattern formation in argentine ants (*Linepithema Humile*). PLoS Comput. Biol. **8** (2012). https://doi.org/10.1371/journal.pcbi.1002592
44. Resnick, M.: Turtles, Termites, and Traffic Jams - Explorations in Massively Parallel Microworlds. MIT Press (1998)

45. Sen, K., Viswanathan, M., Agha, G.: Statistical Model Checking of Black-Box Probabilistic Systems. In: Alur, R., Peled, D.A. (eds.) CAV. LNCS, vol. 3114, pp. 202–215. Springer (2004). https://doi.org/10.1007/978-3-540-27813-9_16
46. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a SAT-Solver. In: FMCAD. LNCS, vol. 1954, pp. 108–125. Springer (2000). https://doi.org/10.1007/3-540-40922-X_8
47. Sumpter, D.J., Blanchard, G.B., Broomhead, D.S.: Ants and agents: A process algebra approach to modelling ant colony behaviour. Bull. Math. Biol. **63** (2001). https://doi.org/10.1006/bulm.2001.0252
48. Theraulaz, G., Bonabeau, E., Nicolis, S.C., Solé, R.V., Fourcassié, V., Blanco, S., Fournier, R., Joly, J.L., Fernández, P., Grimal, A., Dalle, P., Deneubourg, J.L.: Spatial patterns in ant colonies. PNAS **99** (2002). https://doi.org/10.1073/pnas.152302199
49. Tofts, C.M.N.: A synchronous calculus of relative frequency. In: International Conference on Concurrency Theory (CONCUR). LNCS, vol. 458, pp. 467–480. Springer (1990). https://doi.org/10.1007/BFb0039078
50. Tofts, C.M.N.: Describing social insect behaviour using process algebra. Transaction of the Society for Computer Simulation **9**, 227 (1992)
51. Tofts, C.M.N.: Process algebra as modelling. Electr. Notes Theor. Comput. Sci. **162** (2006). https://doi.org/10.1016/j.entcs.2005.12.114
52. Traniello, J.F.A.: Foraging Strategies of Ants. Annual Review of Entomology **34**(1), 191–210 (1989). https://doi.org/10.1146/annurev.en.34.010189.001203
53. Vittori, K., Talbot, G., Gautrais, J., Fourcassié, V., Araújo, A.F.R., Theraulaz, G.: Path efficiency of ant foraging trails in an artificial network. J. theor. Biol. **239** (2006). https://doi.org/10.1016/j.jtbi.2005.08.017
54. Wilensky, U.: Modeling nature's emergent patterns with multi-agent languages. In: EuroLogo (2001)