Analysing collective adaptive systems by proving theorems

Cosimo Perini Brogi, Marco Maggesi

To appear on ISoLA 2024 - Lecture Notes in Computer Science - Springer
https://2024-isola.isola-conference.org/

Analysing collective adaptive systems by proving theorems

Cosimo Perini Brogi[1][0000−0001−7883−5727]
Marco Maggesi[2][0000−0003−4380−7691]

[1] IMT School for Advanced Studies Lucca
cosimo.perinibrogi@imtlucca.it
[2] University of Florence, Italy
marco.maggesi@unifi.it

For Rocco, on the occasion of his 70th birthday[⋆]

Abstract. Inspired by Rocco De Nicola and colleagues' novel approach to the compositional analysis of complex adaptive systems, we foresee an integrated methodology combining those methods with the logical verification techniques offered by modern proof assistants. We explain our long-term perspective on rigorous analysis of ensembles based on these tools for computerised mathematics and propose some preliminary results to make our methodological viewpoint more concrete.

Keywords: Collective adaptive systems · Formal methods · Emergent properties · Logical verification · Interactive theorem proving.

1 Introduction

Humans are familiar with complex adaptive systems (CASs) without realising it. Our Stone Age ancestors witnessed these systems by observing ant colonies at work and the flight of flocks.

Today, we encounter CASs in road traffic, in colonies of microalgae and bacteria in the laboratory, and in cyber-physical distributed systems [47,6,21]. We can study, utilise, and predict them (with sufficient precision) through the tools offered by mathematics and programming languages.

Nevertheless, it remains challenging to assert that we have developed a unified language or a uniform methodology to tackle the challenges posed by these systems. Despite our substantial progress in this field, it is easy to feel like the character created by Italo Calvino [10] between 1975 and 1983, when the study of complex and self-organising systems was at its dawn:

> Se si sofferma per qualche minuto a osservare la disposizione degli uccelli uno in rapporto all'altro, il signor Palomar si sente preso in una

[⋆] We are grateful to Rocco for introducing (via discussions, reading lists, and lots of jokes) the first author – and, indirectly, the second, too – to the field of collective adaptive systems and their rigorous analysis.

> trama la cui continuità si estende uniforme e senza brecce, come se anche lui facesse parte di questo corpo in movimento composto di centinaia e centinaia di corpi staccati ma il cui insieme costituisce un oggetto unitario, come una nuvola o una colonna di fumo o uno zampillo, qualcosa cioè che pur nella fluidità della sostanza raggiunge una sua solidità nella forma. Ma basta che egli si metta a seguire con lo sguardo un singolo pennuto perché la dissociazione degli elementi riprenda il sopravvento ed ecco che la corrente da cui si sentiva trasportato, la rete da cui si sentiva sostenuto si dissolvono e l'effetto è quello d'una vertigine che lo prende alla bocca dello stomaco.[3]

To put it more prosaically, when we wish to study CASs, we find ourselves subjected to two different tensions.

On the one hand, we can harness the power of sophisticated mathematics such as statistical mechanics [45,4] and the theory of differential equations [41], or, from the computational side, the flexibility of architecture description languages [44]. In doing so, we would obtain large and robust models of the system as an autonomous unit. On the other hand, we can also resort to a meticulous and precise formal description of the parts that make up the CAS, its agents or its significant subsets [40,5,43]. This way, we would focus on the formal modelling of local properties and interactions, subsequently exploring the evolution of the complex behaviours through tools aimed at multi-agent system analysis and simulation.

In the first case, we perform a more or less marked abstraction on the compositional nature of CASs, benefiting from the mathematical rigour of the model obtained. This rigour and reliability derive from the robustness of the mathematical theories used, which have been tested for centuries in studying natural and artificial systems. In the second case, we aim for a very natural modelling of CASs, which places the unitary aspects of collective dynamics in the background. These aspects are explored subsequently through formalisation with computational tools for the simulation and verification of large-scale systems.

In this paper, we propose for the first time the use of mathematical logic, and modern proof assistants in particular, as a point of contact between rigorous and natural modelling of complex adaptive systems. We also discuss the possibility of using these tools for computerised mathematics as a unified working environment for the mathematical formalisation, simulation, and formal verification of

[3] Translated in [9] as: "If he lingers for a few moments to observe the arrangement of the birds, one in relation to another, Mr. Palomar feels caught in a weft whose continuity extends, uniform and without rents, as if he, too, were part of this moving body composed of hundreds and hundreds of bodies, detached, but together forming a single object, like a cloud or a column of smoke or a jet of water—something, in other words, that even in the fluidity of its substance achieves a formal solidity of its own. But he has only to start following a single bird with his gaze and the disassociation of the elements returns; and the current that he felt transporting him, the network that he felt sustaining him, dissolve; the effect is that of a vertigo that grips him at the pit of the stomach."

ensembles to complement and, possibly, integrate with the techniques already available to the scientific community.

In the following pages, we will endeavour to elucidate our proposal by addressing some of the potential offered by proof assistants, as well as the challenges posed by their use, for the study of these systems in each of the tasks mentioned above: the development of formal specifications in the language of type theory (Section 3), the simulation of the dynamics of specific systems (Section 4), and the formal verification of the expected emergent properties of generic CASs (Section 5). In Section 6, we showcase some preliminary results to make our arguments more explicit and concrete. We begin this quasi-essay by contextualising our proposal within the literature on the rigorous engineering of complex adaptive systems, acknowledging one methodology as our primary source of inspiration (Section 2). We conclude it (Section 7) with some observations on the reasons for investing in logical verification, even in this area of research.

2 Brief overview of frameworks for ensembles

Various approaches to the rigorous engineering of collective adaptive systems, also known as ensembles, can be found in the literature. Some involve action-based formalisms [3], others use non-classical logics (spatial, temporal, and modal in general) [46], machine learning algorithms [7], and different programming paradigms [22].

The aim of many studies is not only to analyse, model, and use ensembles but also to verify and guarantee that a specific system manifests the expected behaviour. This latter goal is crucial today, as ensembles encompass not only ant colonies or flocks of birds but also smart cities and robot swarms. These systems can have independent and non-communicating components, collaborating (with or without explicit communication) or centrally guided (e.g., by a leader). At the same time, this goal is very challenging for the techniques and tools currently available due to the main defining characteristics of CASs, which are:

- Comprising a potentially large number of distinct entities with individual objectives.
- The ability to adapt dynamically at runtime.
- The emergence of system properties and behaviours that are not directly attributable to the behaviours and properties of individual components (also known as emergent properties).

It is clear, therefore, that the challenge involves not only the development of new tools for studying, programming, and verifying ensembles but also *new methodologies* to achieve success in these objectives.

For example, consider those CASs where the primary system properties emerge through communication (direct or mediated) between individual components. No single "correct" method exists to study this kind of system. It is legitimate to develop a data-driven methodology to predict collective behaviours [7], as well as to work within the paradigm of "aggregate programming"

or "attribute-based programming", abstracting from various characteristics of the components [22,43]. Additionally, purely mathematical models can be used to study these types of ensembles in terms of graphs and more general relational structures, kinetic systems [41], or atemporal causal chains [27,28].

2.1 A different approach

The most recent works by Rocco De Nicola and collaborators propose an additional methodology for studying this type of ensembles [16,17,18,19,20,23]. This approach effectively captures a wide range of examples uniformly using a high-level language based on process algebras. The idea is to define a CAS through its components interacting according to local rules, which involve a distributed data structure formalising biological stigmergy. This methodology is thus inspired by what happens in biological and natural systems, such as ant colonies, to bring about the collective behaviour of interest naturally.

As mentioned earlier, this "bio-inspired approach" requires using precise formal language so that the manifestation of the system's collective properties can be efficiently simulated and verified using robust formal methods. In that work, the language LAbS (Language with Attribute-based Stigmergies)[4] and its operational semantics are introduced and systematically used, along with the SLiVER tool (Symbolic LAbS Verifier)[5] for the automatic translation of LAbS specifications into sequential imperative programs [16,17,24]. These programs are then subjected to simulation and verification tools based on (bounded) model checking and SAT/SMT solvers.

In the present context, we are interested in a specific characteristic of this bottom-up methodology: using a compositional computational structure (the process algebra behind the operational semantics of LAbS) to model and analyse ensembles. This choice has made the formalisation of system specifications particularly intuitive, flexible, and extendable, much more natural to handle than the mathematical models typically used for these systems in fields such as biology. Additionally, it has allowed for a rigorous formal verification of their main properties.

This compositional, bio-inspired, and formally rigorous methodology has significant intrinsic value, already evident in the combination of "LAbS + SLiVER + model checking." This same methodology can also be embodied by different tools derived from contemporary mathematical logic, which in some aspects are complementary to more traditional formal methods, namely type theory-based proof assistants. Currently, we have only some promising elementary examples to support our thesis. However, in the following sections, we will clarify how to translate and instantiate Rocco and collaborators' methodology into computerised mathematics. In the present work, we focus on high-level concepts

[4] The source code for the LAbS code generator for formal specifications of collective systems is freely available from the repository https://github.com/labs-lang/labs.

[5] The source code and binary releases of SLiVER for Linux x64 systems are freely available from the repository https://github.com/labs-lang/sliver.

shared by a potentially large class of formal models, encompassing both Rocco's approach [16,17,20] and our recent results documented in [39].

3 Formalisation

According to the bottom-up methodology we intend to apply, whether we opt to simulate or verify the behaviour of a CAS, we require a formal language to describe the system in terms of its components and the distributed data structure (stigmergy) that facilitates interaction among individual components.

The type theories underpinning modern proof assistants are sufficiently expressive to encapsulate the concept of interactive agents with individual behaviours, local copies of stigmergy, and an evolving environment. We can also effortlessly formalise all the local rules/link predicates of, e.g., [17] that govern the dynamics of the components (and, thus, the collective system). All this can be achieved with a single formal language and a unified logical framework.

Type theory (simple or dependent)[6] permits this because each of these notions – agent, virtual stigmergy, system, environment – is defined through a distinct type within the same formal system. For instance, a single step in the dynamics of an agent is defined as a function—specified in the language of the theory—that takes an element of type 'agent' and an element of type 'stigmergy' and returns a collection of possible attributes of the agent. This step thus defines the individual agent's logic (or behaviour) in functional and formal terms. The system's evolution can similarly be defined as a typed function in terms of individual update steps according to the logic of the component and the dynamics of the stigmergy (also defined in functional terms).

In this manner, type theory provides a high-level computational language akin to LAbS for formalising and describing CASs. It also offers the additional level of abstraction afforded by the operational semantics of LAbS, as it enables us to define a generic type 'agent' through the type variables 'attribute', 'local-knowledge', and 'possible-actions'. Depending on the system under consideration, we can instantiate these three type variables and then concentrate on defining the typed functions that represent the dynamics of individual agents, thereby translating into mathematical language the listings that describe the types of agents considered in, e.g., [20].

4 Simulation

Once a CAS is formalised, the standard bottom-up methodology outlined in [17,20] instructs us to translate the LAbS specifications for the system and the operational semantics of the language into sequential imperative programs, written for example in C, on which the actual simulation takes place.

Using type theory, we can replace all the delicate and ingenious work of sequential emulation detailed in [23] with a conversion function that operates on the typed term defining the system's initialisation state.

[6] Refer to [34,14] for a hands-on introduction.

In addition to basic low-level conversions—such as handling `let...in` constructions within the formal system description—this function will require logical conversions to automatically reformulate the formal analogues of the predicate links that govern communication between agents and other syntactic optimisations.

These latter optimisations depend on the availability, within the proof assistant at hand, of formalised proofs of mathematical theorems that allow for the gradual refinement of specifications from their original formulation—usually easy for a human user to read—into a form better managed by the remaining generic conversion operations and automatic reasoning processes that are part of the proof assistant's logical kernel.

Simulation thus becomes the straightforward automatic execution of a call-by-value evaluation of the evolution function—describing the behaviour (or logic) of the components/agents—on the typed terms corresponding to the system's initial state (and the subsequent possible steps in its evolution).

In our context, the non-deterministic component of the agents' dynamics is managed by set-theoretic concepts and operations. These replace the scheduler used in the conventional context's sequential emulation of LAbS specifications. This approach guarantees the correctness of the simulation concerning the intended dynamics, owing to the theorems that have already been formally proven in the proof assistant and utilised in the earlier conversion process.

By applying the bottom-up methodology of Rocco and his collaborators within a formal proof environment, we can avoid translating the formal specifications and their semantics into C programs for simulation. Instead, we execute and simulate the specifications directly within the proof assistant. These specifications are already articulated in a typed functional programming language, or more precisely, in the language of the logical engine that underpins the proof assistant.

However, it is fair to recognise that reducing simulation to an extensive call-by-value evaluation could be more efficient. Moreover, this approach forfeits the advantages of advanced techniques available for analysing C programs, as detailed in works such as [48,12,38,30].

5 Verification

Verification and simulation are complementary approaches in the analysis of ensembles. Simulation is more efficient for empirically testing the potential emergence of expected collective behaviour. Formal verification, while more computationally intensive, provides a stronger guarantee of the emergence of this behaviour or absence thereof. It is not uncommon for verification to identify specific system dynamics that simulation might miss, as noted in [19].[7]

[7] This discrepancy between the outcomes of simulation and verification is not surprising to those familiar with software quality and safety using formal methods. A rigorously verified bug always invalidates any successful simulations of the program's functional behaviours under examination.

The standard bottom-up methodology includes automatic formal methods for verifying ensembles. The SLiVER tool uses the same backend technique for both simulation and verification [17,24,23]. Verifying a CAS's behaviour is reduced to reachability analysis of the corresponding non-deterministic sequential imperative program, also used in the simulation process.

Similarly, a proof assistant can simulate the system's evolution and *prove* properties of the CAS. Through conversions, we can simulate the system's evolution. Moreover, we can state the property we intend to verify in the language of type theory. This property becomes a *mathematical theorem*, ready to be formally proven within the interactive proof environment of the proof assistant, using the inference rules of the same underlying type theory.

In this way, the proof assistant does provide a unified workspace for constructing a model, writing a specification, simulating the system's dynamics, and mathematically proving the expected property. It also allows for the identification of potential counterexamples and rare events.

It must be acknowledged that while this methodological homogeneity and mathematical rigour are significant gains, there is a considerable loss in automation. In the standard version (LAbS+SLiVER) of the bottom-up methodology, simulation and verification may require lengthy execution times for the automatic tools. The user waits for the result without active engagement during these processes. In our proposed methodology, verification demands varying degrees of user participation, as the user must develop a formal proof according to the language and rules of the proof assistant. This workload cannot be reduced or compared to the push-button approach more prevalent in formal methods.

However, our proposal offers added value as it is less affected by issues related to the size of ensembles. Reducing the verification of a CAS property to a reachability or model-checking problem exposes the process to well-known issues related to state explosion, partially addressed by current parameterised model-checking techniques. Moreover, model checking-based verification can only operate on a fixed-size model. This limitation restricts consideration to ensembles with a predetermined number of agents, even when the property of interest is simple to specify and independent of system size, as in the example of colliding ants studied in [20].

To achieve a tool capable of unbounded verification, the current model checking-based workflow appears to require integration with different program analysis methods, such as k-induction [49], completeness thresholds [15], or property-directed analysis [50].

On the other hand, when working with a proof assistant, it is relatively straightforward to reason about ensembles of arbitrary sizes to demonstrate whether a property is satisfied, generalising over predetermined dimensional parameters.

Furthermore, the expressiveness of type theory allows for the formalisation of any collective property one might expect from the CAS, including those typically beyond the scope of model checking logics like LTL and CTL [31].

6 Make it concrete

In this section, we present two different approaches to analysing an elementary ensemble, specifically a colony of foraging ants depicted in the idealised scenario of Figure 1

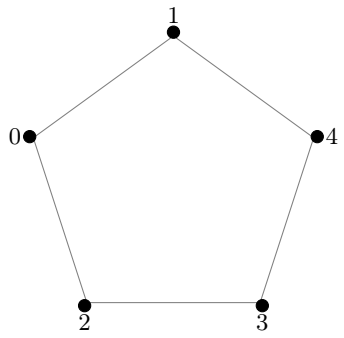

Fig. 1. The discrete model for the idealised version of the double bridge experiment

The ant environment is represented by a simple graph resembling a regular pentagon, with the node labelled 0 serving as the nest and the node labelled 4 as the food source. Ants move discretely between adjacent nodes in unit steps, with the ratio r between the path 0–1–4 and 0–2–3–4 being 2, akin to the laboratory experiment detailed in [32]. They deposit pheromones on intermediate nodes and opt for the shorter or longer path based on higher pheromone concentration when at the nest or the food source.

Biologists experimenting with the colony have observed an emergent property: the gradual convergence of foraging ants towards the shortest path 0–1–4. This serves as an illustration of emergent self-organisation, guided by indirect communication through pheromone release and detection, a prime example of stigmergic interaction among agents.[8]

6.1 Ants in an SMT-solver

Let us initially examine the system described above using the language and capabilities of the SMT-checker. In our experiments, we employed Z3, a widely utilised solver in the community of formal methods and program analysis [42].[9]

A natural approach to formalise our scenario is as a transition system, linking the system state at a given time t with the subsequent system state at $t + 1$.

[8] Refer to [37,32,25,20] for a more detailed discussion of this phenomenon.

[9] We survey here some snippets of code, which is freely available from our online notebook, also archived on Software Heritage.

Simulation of the dynamics then simplifies to a relatively straightforward constraint satisfaction problem. Similarly, verifying the emergent behaviour of the foraging colony entails certifying that the negation of the thesis – namely, the convergence of the ants on the shortest path – is unsatisfiable.

By relying on (potentially refined versions of) this technology, our focus can be directed towards precisely describing the specific colony under consideration and formalising the system specifications. Simulation and verification then become effortlessly achievable, aligning with the push-button methodology of automated model checking. Nonetheless, automation comes at a cost in terms of generality and expressive capabilities. Both simulation and verification of expected properties can only be conducted on specific instances of the idealised model – namely, on colonies of predetermined size within the specification and for evolutions with a pre-set time bound – due to the reliance on first-order logic alone.

6.2 Ants in Higher-Order Logic

Our central assertion has been this: in principle, a proof assistant is a unified platform for all the mathematical tasks involved in analysing a CAS, encompassing formal specification, automated simulation, and rigorous verification.

We have explored this potential in a related study [39], and now summarise our findings concerning the idealised foraging colony analysed within the HOL Light proof assistant.[10]

A simulation for an ant colony of fixed size can be implemented as an automated call-by-value evaluation of the evolution function within HOL Light. This function is executed iteratively to generate a complete collection of potential states arising from the colony's initial configuration.[11]

Moreover, we can *abstract from* the specific parameters of the ant colony – most notably, *its size* – and proceed with rigorous verification to confirm that the expected convergence on the shortest path occurs whenever certain explicit conditions are met, independently of the number of ants involved in the dynamics. This emergence is formulated as a *formal theorem* in the language of type theory and *formally proven* irrespective of the number of ants comprising the colony.

6.3 Finding the right path

In our technical paper [39], we have shown how to model, simulate, and verify *for colonies of arbitrary size* the convergence of foraging ants on the shortest

[10] For the source code, please refer to our GitHub repository HOL-Ants, containing the formal analysis discussed here. The same code is also archived on Software Heritage.

[11] Although the efficiency of this evaluation can be improved, our current demonstration suffices to show that exploration of the system dynamics can indeed be performed within the proof assistant on a mid-level personal computer, without resorting to external resources.

path within a discrete, abstract, and idealised environment, only using the proof assistant HOL Light [35]. Enhancing the performance of the conversion function used to simulate the long-term dynamics of colonies of fixed size is possible. This improvement can be achieved within the proof assistant itself and may lead to performances comparable to the efficiency of the SMT-based model.

On the contrary, due to the intrinsic limitations of the formal language and engineerisation of SMT-solvers, it is probably impossible to verify the dynamics of colonies of arbitrary size, even though such a task for a specific colony can be (quite easily) accomplished automatically after fixing the number of ants within the model.

These first experiments confirm that, even for such a simple scenario, there exists a trade-off between the expressive capability and exactitude of proof assistants, on the one hand, and the optimised automation and ease of use of SMT-solvers. Notice that the methodologies employed in the two proposed experiments are so distinct that a quantitative comparison is challenging, if only partially meaningful, at the current stage of development.

Future advances in interactive theorem proving could shorten the difference between these two approaches. Furthermore, we foresee a fruitful integration of the two technologies to distribute the workflow between a formal platform optimised for the exploration of dynamics (i.e., the automated theorem prover) and one specialised in the rigorous certification of the expected properties of these dynamics (i.e., the proof assistant).

Our perspective prioritises using highly expressive general systems such as HOL Light or other proof assistants. We know there are tools, like the Dafny language [51], which integrate proof techniques based on SAT/SMT. However, our current interest lies in starting with general tools that can subsequently be specialised to make them practical for a wide range of interesting cases.

At the same time, we recognise the importance of providing end-users with intuitive and user-friendly tools. We understand that an intuitive environment for formal proof development is crucial for widespread adoption, particularly for researchers who may need to become more familiar with computer-aided mathematics. To address this, we are considering the development of extensions and customisations of HOL Light to facilitate its use in such contexts.

7 Conclusions

Proof assistants are computer programs that enable us to perform mathematics with the highest level of precision available. Their expressive capacity and formal exactness find a natural application in software and hardware verification, in industrial and academic research as well [33,36,26,2,14,11].

Over the past years [29], they have made an essential contribution to the construction and discovery of new areas of mathematics and their rigorous structuring [8,1]. Moreover, they are proving to be exceptional tools for boosting a long-awaited process of renewing peer-review procedures and for democratising the communication of scientific results [13].

In this methodological paper, we have suggested how to use these tools for the rigorous engineering of collective adaptive systems, their analysis, and, most importantly, the mathematical verification of their expected behaviours. Our proposal is inspired by an existing bottom-up approach to ensemble modelling [20], which we propose to translate into a framework focused on the use of proof assistants for every essential aspect of the analysis (specification, simulation, and verification).

We have outlined our long-term perspective on the feasibility of such a unified methodology and the challenges to face, presenting some initial positive results in this direction. There remains ample scope for further work and refinement, particularly concerning the efficiency of a comprehensive analysis of CASs centred on proof assistants. Nonetheless, this endeavour holds scientific significance within a broader horizon, imbued with more general philosophical implications. In a sense, with our proposal, we aim to introduce new tools and methods to precisely understand the phenomena involving ensembles, preceding the use of new technologies based on approximate knowledge of these systems. In literary terms, the arguments we have presented in the preceding pages may also reveal a personal affinity for Mr Palomar's cognitive effort and our desire to alleviate, through the precision offered by mathematical logic, that sense of vertigo described at the paper's outset – a sensation that has likely accompanied (some members of) our species since time immemorial.

Acknowledgments. We thank two anonymous reviewers for their comments and feedback on the first submission of this work; the current version definitely gained from their suggestions regarding clarity and readability.

This work was partially funded by: the MIUR project PRIN 2017FTXR7S IT-MATTERS (Methods and Tools for Trustworthy Smart Systems); the MIUR project PRIN 2017JZ2SW5 "Real and Complex Manifolds: Topology, Geometry and holomorphic dynamics"; the project SERICS PE00000014 (SEcurity and RIghts in the CyberSpace) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU (MUR Code: 2022CY2J5S); Istituto Nazionale di Alta Matematica – INdAM group GNSAGA.

Disclosure of Interests. The authors have no competing interests to disclose.

References

1. Avigad, J.: Mathematics and the formal turn. Bulletin (New Series) of the American Mathematical Society **61**(2) (2024). https://doi.org/https://doi.org/10.1090/bull/1832
2. Baanen, A., Bentkamp, A., Blanchette, J., Hölzl, J., Limperg, J.: The Hitchhiker's Guide to Logical Verification (2024)
3. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
4. 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). https://doi.org/10.1073/pnas.0711437105, https://www.pnas.org/doi/abs/10.1073/pnas.0711437105

5. Beal, J., Viroli, M.: Aggregate programming: From foundations to applications. Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems: 16th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2016, Bertinoro, Italy, June 20-24, 2016, Advanced Lectures 16 pp. 233–260 (2016)
6. Bortolussi, L., De Nicola, R., Gast, N., Gilmore, S., Hillston, J., Massink, M., Tribastone, M.: A quantitative approach to the design and analysis of collective adaptive systems. In: 1st FoCAS Workshop on Fundamentals of Collective Adaptive Systems (2013)
7. Bureš, T., Hnětynka, P., Kruliš, M., Plášil, F., Khalyeyev, D., Hahner, S., Seifermann, S., Walter, M., Heinrich, R.: Generating adaptation rule-specific neural networks. International Journal on Software Tools for Technology Transfer **25**(5), 733–746 (2023)
8. Buzzard, K.: Mathematical reasoning and the computer. Bulletin (New Series) of the American Mathematical Society **61**(2) (2024). https://doi.org/https://doi.org/10.1090/bull/1836
9. Calvino, I.: Mr Palomar. Vintage classics, Vintage (1994), English translation of [10] by W. Weaver
10. Calvino, I.: Palomar. Mondadori (2013)
11. Chapman, R., Petcher, A., Hansen, T., Peng, Y., Lepoint, T., Bytheway, C., Kampanakis, P.: Formal Verification of Cryptographic Software at AWS: Current Practices and Future Trends. nist.org (2024)
12. Chen, H.Y., David, C., Kroening, D., Schrammel, P., Wachter, B.: Synthesising interprocedural bit-precise termination proofs (T). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 53–64 (2015). https://doi.org/10.1109/ASE.2015.10
13. Cheng, E.: How machines can make mathematics more congressive. Bulletin (New Series) of the American Mathematical Society **61**(2) (2024). https://doi.org/https://doi.org/10.1090/bull/1827
14. Chlipala, A.: Certified programming with dependent types: a pragmatic introduction to the Coq proof assistant. MIT Press (2022)
15. Clarke, E., Kroening, D., Ouaknine, J., Strichman, O.: Completeness and complexity of bounded model checking. In: Steffen, B., Levi, G. (eds.) Verification, Model Checking, and Abstract Interpretation. pp. 85–96. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
16. De Nicola, R., Di Stefano, L., Inverso, O.: Toward formal models and languages for verifiable multi-robot systems. Frontiers Robotics AI **5**, 94 (2018). https://doi.org/10.3389/FROBT.2018.00094, https://doi.org/10.3389/frobt.2018.00094
17. De Nicola, R., Di Stefano, L., Inverso, O.: Multi-agent systems with virtual stigmergy. Sci. Comput. Program. **187**, 102345 (2020). https://doi.org/10.1016/J.SCICO.2019.102345, https://doi.org/10.1016/j.scico.2019.102345
18. De Nicola, R., Di Stefano, L., Inverso, O., Valiani, S.: Modelling flocks of birds from the bottom up. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Adaptation and Learning - 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22-30, 2022, Proceedings, Part III. Lecture Notes in Computer Science, vol. 13703, pp. 82–96. Springer (2022). https://doi.org/10.1007/978-3-031-19759-8_6, https://doi.org/10.1007/978-3-031-19759-8_6
19. De Nicola, R., Di Stefano, L., Inverso, O., Valiani, S.: Intuitive modelling and formal analysis of collective behaviour in foraging ants. In: Pang, J., Niehren, J.

(eds.) Computational Methods in Systems Biology - 21st International Conference, CMSB 2023, Luxembourg City, Luxembourg, September 13-15, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14137, pp. 44–61. Springer (2023). https://doi.org/10.1007/978-3-031-42697-1_4, https://doi.org/10.1007/978-3-031-42697-1_4

20. 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. **25**(5), 675–691 (2023). https://doi.org/10.1007/S10009-023-00731-0, https://doi.org/10.1007/s10009-023-00731-0
21. De Nicola, R., Jähnichen, S., Wirsing, M.: Rigorous engineering of collective adaptive systems. International Journal on Software Tools for Technology Transfer **22**, 389–397 (2020)
22. De Nicola, R., Loreti, M., Pugliese, R., Tiezzi, F.: A formal approach to autonomic systems programming: the scel language. ACM Transactions on Autonomous and Adaptive Systems (TAAS) **9**(2), 1–29 (2014)
23. Di Stefano, L., De Nicola, R., Inverso, O.: Verification of distributed systems via sequential emulation. ACM Trans. Softw. Eng. Methodol. **31**(3), 37:1–37:41 (2022). https://doi.org/10.1145/3490387, https://doi.org/10.1145/3490387
24. Di Stefano, L., Lang, F., Serwe, W.: Combining SLiVER with CADP to analyze multi-agent systems. In: Bliudze, S., Bocchi, L. (eds.) Coordination Models and Languages - 22nd IFIP WG 6.1 International Conference, COORDINATION 2020, Held as Part of the 15th International Federated Conference on Distributed Computing Techniques, DisCoTec 2020, Valletta, Malta, June 15-19, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12134, pp. 370–385. Springer (2020). https://doi.org/10.1007/978-3-030-50029-0_23, https://doi.org/10.1007/978-3-030-50029-0_23
25. Dorigo, M., Stützle, T.: Ant colony optimization: overview and recent advances. Springer (2019)
26. Ferguson, W.E., Bingham, J., Erkök, L., Harrison, J.R., Leslie-Hurd, J.: Digit serial methods with applications to division and square root. IEEE Transactions on Computers **67**(3), 449–456 (2017)
27. Fettke, P., Reisig, W.: Discrete models of continuous behavior of collective adaptive systems. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Adaptation and Learning - 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22-30, 2022, Proceedings, Part III. Lecture Notes in Computer Science, vol. 13703, pp. 65–81. Springer (2022). https://doi.org/10.1007/978-3-031-19759-8_5, https://doi.org/10.1007/978-3-031-19759-8_5
28. Fettke, P., Reisig, W.: A causal, time-independent synchronization pattern for collective adaptive systems. Int. J. Softw. Tools Technol. Transf. **25**(5), 659–673 (2023). https://doi.org/10.1007/S10009-023-00733-Y, https://doi.org/10.1007/s10009-023-00733-y
29. Fraser, M., Granville, A., Harris, M.H., McLarty, C., Riehl, E., Venkatesh, A.: Will machines change mathematics? Bulletin (New Series) of the American Mathematical Society **61**(2) (2024). https://doi.org/https://doi.org/10.1090/bull/1833
30. Gadelha, M.R., Monteiro, F.R., Morse, J., Cordeiro, L.C., Fischer, B., Nicole, D.A.: ESBMC 5.0: an industrial-strength C model checker. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. p. 888–891. ASE '18, Association for Computing Machinery, New York, NY,

USA (2018). https://doi.org/10.1145/3238147.3240481, https://doi.org/10.1145/3238147.3240481
31. Goranko, V., Rumberg, A.: Temporal Logic. In: Zalta, E.N., Nodelman, U. (eds.) The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University, Summer 2024 edn. (2024), https://plato.stanford.edu/archives/sum2024/entries/logic-temporal/
32. Goss, S., Aron, S., Deneubourg, J.L., Pasteels, J.M.: Self-organized shortcuts in the argentine ant. Naturwissenschaften **76**(12), 579–581 (1989)
33. Harrison, J.: Floating-point verification. In: Fitzgerald, J., Hayes, I.J., Tarlecki, A. (eds.) FM 2005: Formal Methods, International Symposium of Formal Methods Europe, Proceedings. Lecture Notes in Computer Science, vol. 3582, pp. 529–532. Springer-Verlag (2005)
34. Harrison, J.: HOL Light tutorial. http://www.cl.cam.ac.uk/~jrh13/hol-light/tutorial.pdf (2017)
35. Harrison, J.: The HOL Light Theorem Prover. Available at https://github.com/jrh13/hol-light (2024)
36. Harrison, J., Urban, J., Wiedijk, F.: History of Interactive Theorem Proving. In: Computational Logic. vol. 9, pp. 135–214 (2014)
37. Hölldobler, B., Wilson, E.O.: The Ants. Belknap Press of Harvard University Press (1990), https://books.google.it/books?id=R-7TaridBX0C
38. Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-c: A software analysis perspective. Form. Asp. Comput. **27**(3), 573–609 (may 2015). https://doi.org/10.1007/s00165-014-0326-7, https://doi.org/10.1007/s00165-014-0326-7
39. Maggesi, M., Perini Brogi, C.: Rigorous analysis of idealised pathfinding ants in higher-order logic. ISoLA 2024 (This issue), Lecture Notes in Computer Science, Springer (2024), HAL preprint hal-04620418
40. Mefteh, W., Migeon, F., Gleizes, M.P., Gargouri, F.: ADELFE 3.0: Design, building adaptive multi agent systems based on simulation. A case study. In: Computational Collective Intelligence: 7th International Conference, ICCCI 2015, Madrid, Spain, September 21-23, 2015, Proceedings, Part I. pp. 19–28. Springer (2015)
41. Monica, S., Bergenti, F., Zambonelli, F.: A kinetic approach to investigate the collective dynamics of multi-agent systems. International Journal on Software Tools for Technology Transfer **25**(5), 693–705 (2023)
42. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings. Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24, https://doi.org/10.1007/978-3-540-78800-3_24
43. Murgia, M., Pinciroli, R., Trubiani, C., Tuosto, E.: Comparing perfomance abstractions for collective adaptive systems. International Journal on Software Tools for Technology Transfer **25**(5), 785–798 (2023)
44. Ozkaya, M., Kloukinas, C.: Are we there yet? Analyzing architecture description languages for formal analysis, usability, and realizability. In: 2013 39th Euromicro Conference on Software Engineering and Advanced Applications. pp. 177–184. IEEE (2013)
45. Parisi, G.: Nobel Lecture: Multiple equilibria. Rev. Mod. Phys. **95**, 030501 (Aug 2023). https://doi.org/10.1103/RevModPhys.95.030501, https://link.aps.org/doi/10.1103/RevModPhys.95.030501

46. Platzer, A.: The logical path to autonomous cyber-physical systems. In: International Conference on Quantitative Evaluation of Systems. pp. 25–33. Springer (2019)
47. Priami, C., Quaglia, P.: Global Computing: IST/FET International Workshop, GC 2004, Rovereto, Italy, March 9-12, 2004, Revised Selected Papers, vol. 3267. Springer (2005)
48. Qadeer, S., Wu, D.: Kiss: keep it simple and sequential. SIGPLAN Not. **39**(6), 14–24 (jun 2004). https://doi.org/10.1145/996893.996845, https://doi.org/10.1145/996893.996845
49. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a sat-solver. In: Hunt, W.A., Johnson, S.D. (eds.) Formal Methods in Computer-Aided Design. pp. 127–144. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
50. Shemer, R., Gurfinkel, A., Shoham, S., Vizel, Y.: Property directed self composition. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification. pp. 161–179. Springer International Publishing, Cham (2019)
51. Sitnikovski, B.: Introducing Software Verification with Dafny Language: Proving Program Correctness. Apress (2022)