Analysing Collective Adaptive Systems by Proving Theorems

Cosimo Perini Brogi° Marco Maggesi*

°IMT School for Advanced Studies Lucca
*University of Florence

REoCAS Colloquium @ ISoLA, celebrating Rocco De Nicola's 70th Birthday
Crete, 26-31 October 2024

Work supported by the project SERICS - PE0000014, financed within PNRR, M4C2 I.1.3, funded by the European Union - NextGenerationEU

What are CASs?

- Systems composed of interacting *agents* that collectively adapt and evolve based on *local* interactions.

Several different approaches

- Statistical Physics: Apply solid mathematical theories and techniques to model system evolution.
- Non-Classical Logics: Use temporal and modal logics for expressing and verifying system properties.
- Action-based Formalisms: Focus on capturing the dynamics of individual agents.
- Machine Learning Algorithms: Predict and model collective behaviour on the basis of probabilistic dynamics.
- Various Programming Paradigms: Abstract from individual properties at different levels, with more or less rigorous semantics.
- …
- **Rocco's way:** Use process algebras, where each agent is a process running in parallel and *indirectly* interacting with each other component of the system

Searching for a unified framework

Mathematical Models:

Robust, large-scale view of CASs as autonomous systems, focused on the whole ensemble

Formal Descriptions for Programming:

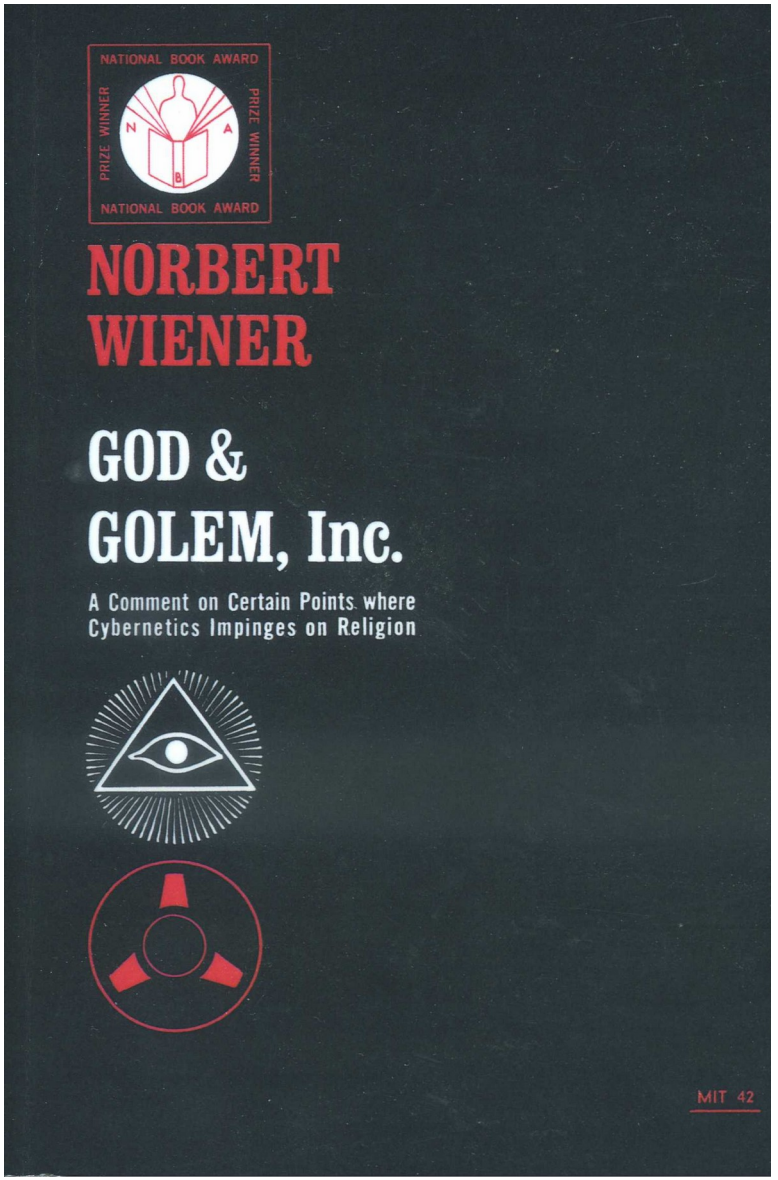
Local properties and interactions among agents are explored via multi-agent system tools for simulation and verification

Our proposal

Use proof assistants to model, simulate, and logically verify the behaviour of CASs within a single formal framework.

Some benefits of the proposed methodology

- ✔ Highest level of precision in specifications
- ✔ Rigorous formalisation and simulation
- ✔ Emergent system behaviours are *formally proven* to manifest out of the agents' logic/dynamics as mathematical theorems



Challenging some inclinations

The big red button

Good looking code is not always the best solution: programmers are not gods, technology is not faith

Learn from experience:

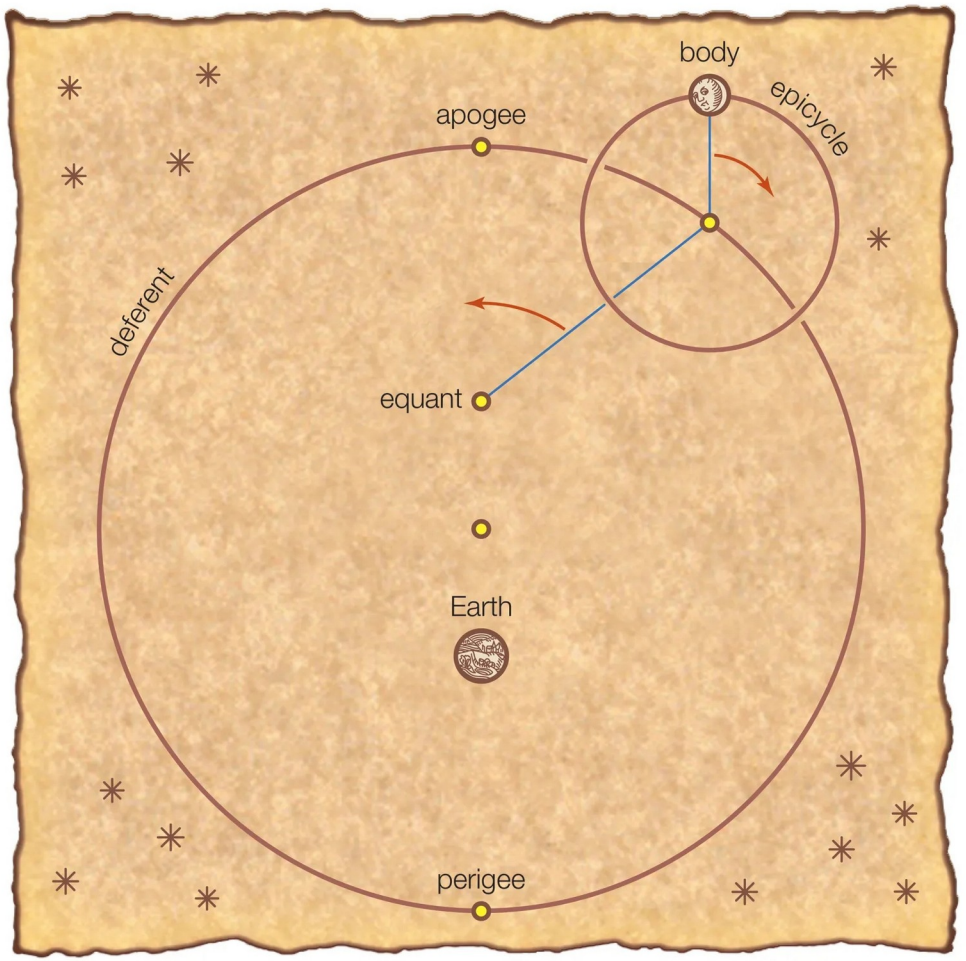
- ✗ Error in floating-point division instructions on some Pentium processors
- ➔ $475m to cover the costs.

- ✗ Software bug in the Ariane 5's Inertial Reference System (64-bit floating variable into a 16-bit integer)
- ➔ $370m wasted in launch, huge project delay, bad reputation

Challenging some inclinations

The model of models

"All models are wrong.
Some are wronger"



Challenging some inclinations

Down with formalisms

You are shown a set of four cards placed on a table, each of which has a number on one side and a colour on the other. The visible faces of the cards show “3”, “8”, “blue” and “red”. Which card(s) must you turn over in order to test that if a card shows an even number on one face, then its opposite face is blue?

You are shown a set of four cards placed on a table, each of which has an age on one side and a drink on the other. The visible faces of the cards show "drinking alcohol", "drinking soda", "16 years old", and "25 years old". Which card(s) must you turn over to test the rule that if someone is drinking alcohol, then they must be 18 or older?

Why proof assistants

Definition

A piece of software for developing mathematical proofs about mathematical structures, programs and their formal specifications, and checking the correctness of these proofs *using the computer*.

- Trusting a theorem or a piece of code means now trusting the formal mathematical theory underlying the proof assistant one uses to write statements, proofs, programs, and specifications.
- The implementation of that theory can itself be machine checked
- Each step of your modelling, reasoning, and analysis is certified to be correct by construction
- → For CASs, we unify modelling and verification, with no explicit need for separate simulation programs

Type theoretic formalisation

A type-theoretic definition of non-deterministic agents in CASs:

```
(*                                                                          *)
(* Descriptions of the type parameters:                                     *)
(*   A = internal status of the agent                                       *)
(*   B = perception                                                         *)
(*       (information gathered by the agent about the surrounding env)      *)
(*   C = action (possible choices made by the agent)                        *)
(* ------------------------------------------------------------------------ *)

let input_INDUCT,input_RECUR = define_type
  "input = Input(A#B)";;

let INPUT_STATUS = define
  `INPUT_STATUS (Input(stat:Stat,percpt:Percpt)) = stat`;;

let INPUT_PERCEPTION = define
  `INPUT_PERCEPTION (Input(stat:Stat,percpt:Percpt)) = percpt`;;

add_ants_thl[injectivity "input"; INPUT_STATUS; INPUT_PERCEPTION];;

let agent_INDUCT,agent_RECUR = define_type
  "agent = Agent((A,B)input->A#C->bool)";;

let AGENT_STEP = define
  `AGENT_STEP (Agent(a:(Stat,Percpt)input->Stat#Action->bool)) = a`;;

add_ants_thl[injectivity "agent"; AGENT_STEP];;
```

- Agents are defined by a tuple of attributes of specific types
- Their behaviour/logic is a computable function of the PA mapping attributes values into attributes values

Type theoretic formalisation

A type-theoretic definition of the system

```
(*
A = num->num option - Environment (stigmergy) of the whole system
B = num - Ident
C = num - Agent attribute (position)

D = direction - Agent internal status
E = Agent perception
F = (num#direction) - new agent status after agent action
*)

let system_INDUCT,system_RECUR = define_type
  "system = System((num->num option)#
                   (num,num#direction#(D,E,F)agent)dict)";;

let SYSTEM_ENVIRONMENT = define
  `SYSTEM_ENVIRONMENT
    (System(env,agents):(Env,Id,Attr,Stat,Percpt,Action)system) =
    env`;;

let SYSTEM_AGENTS = define
  `SYSTEM_AGENTS
    (System(env,agents):(Env,Id,Attr,Stat,Percpt,Action)system) =
    agents`;;

add_ants_thl[injectivity "system"; SYSTEM_ENVIRONMENT; SYSTEM_AGENTS];;

let UPDATE_SYSTEM = new_definition
  `UPDATE_SYSTEM
    (update_environment :
      (Env,Id,Attr,Stat,Percpt,Action)system -> Env)
    (update_agents :
      (Env,Id,Attr,Stat,Percpt,Action)system ->
      (Id,Attr#Stat#(Stat,Percpt,Action)agent)dict -> bool)
    (sys : (Env,Id,Attr,Stat,Percpt,Action)system) :
    (Env,Id,Attr,Stat,Percpt,Action)system -> bool =
    IMAGE (\a. System(update_environment sys,a))
          (update_agents sys)`;;
```

- A system is a tuple of agents and environmental information
- The system dynamics is a computable function defined in terms of the logic of agents, mapping a system configuration into a set of possible next configurations

Simulation using proof assistants

- The process algebraic approach needs highly ingenious techniques to translate formal specifications of a CAS into C programs, so that emergence of the collective property is simulated as a reachability statement of the programs corresponding to the specifications

- After formalising the model and the dynamics in a type-based PA, we can perform within the PA a direct simulation of the system dynamics as the latter is defined as a functional program evaluated by the machine

  Pros: No external translation is needed, correctness of the output is guaranteed by a certified computation on any mid-level computer
  Cons: Naive computations are not efficient, implementation/compilation of the functional program can be time consuming

Verification using proof assistants

Model checking is known to struggle with large systems in many situations, and CASs make no exceptions.

- A proof assistant can state in type-theoretic language the system property we need to verify and formally prove that the property holds as a mathematical theorem
- It is not unusual that, during the interactive proof of the property, the user identifies potential counterexamples and rare events in the system dynamics, leading to an improved formalisation and design

Pros: The emergent behaviour is proven to manifest in systems of *arbitrary* size, model-checking escaping properties are easily captured by type theory,
Cons: Limited automation, very active participation required from the human user

Finding the right path

- Proof assistants offer a unique opportunity for developing a **unified and rigorous framework** for CAS analysis (modelling, simulation, verification)
  - ✔ We already have promising results demonstrating PAs' ability to simulate dynamics and verify emergent properties of a simple colony of foraging ants[1]

- They address the limitations of traditional formal methods for large systems, by handling, proving and refuting statements over arbitrary system sizes
  - Current efficiency of functional simulation slightly lags behind traditional tools (SAT/SMT solvers)

- An integration of logical verification based on proof assistants with state-of-the-art automated reasoning tools (e.g. Lean+Z3) opens new horizons for studying collective adaptive systems

1. Maggesi, M., Perini Brogi, C.: *Rigorous analysis of idealised pathfinding ants in higher-order logic*, ISoLA 2024 (talk from two days ago)

Thanks for your attention!

Proof is an idol before whom the pure mathematician tortures himself.
– Arthur Eddington