IMT SCHOOL FOR ADVANCED STUDIES LUCCA

Open Problems in Concurrency Theory

Bertinoro (Italy), 26-30 June 2023

Open Problems in Concurrency Theory

MODELING FLOCKS AND THEIR EMERGENT BEHAVIORS

Rocco De Nicola

Scuola IMT Alti Studi, Lucca, Italy

Joint work with Luca Di Stefano (Univ. Gothenburg), Omar Inverso (GSSI, L'Aquila), Serenella Valiani (Scuola IMT Lucca)

OPCT – Bertinoro June 26 – 30, 2023

Flocking behaviour

- Decentralized system of autonomous components
- Emergent patterns of collective behaviour
  - Coherent movements
  - Cohesion

(video: https://www.quantamagazine.org/smarter-parts-make-collective-systems-too-stubborn-20190226/?mc_cid=1e132cc211&mc_eid=9307f60615)

- https://www.youtube.com/watch?v=3w90X92pDSs

Language-based methodology

Languages play a key role in the engineering of systems

- Systems must be specified as naturally as possible
- Distinctive aspects of the domain need to be first-class citizens
- Intuitive/concise specifications can avoid encodings

Models need to be strictly related to languages for effective analysis

- high-level abstract models guarantee feasible investigations
- scrutiny of results (e.g., counterexample) based on system features, rather than on their low-level representation, guarantees better feedback

Systems Modelling and Analysis

Big challenge: devise appropriate abstractions and linguistic primitives to deal with the specificities of the systems under consideration while relying on an appropriate semantic model.

A possible approach: Combined use of formal methods with model-driven software engineering.

Key ingredients:

1. A specification language equipped with a formal semantics
2. A programming framework with associated runtime environment
3. A number of verification techniques and associated tools

IMT SCHOOL FOR ADVANCED STUDIES LUCCA

Our Contributions: A timeline

Languages and tools for different classes of distributed systems

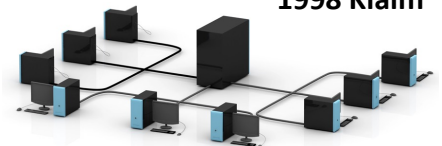
Service-oriented computing
- services composition
- heterogeneous components
- code reuse, interoperability

2006-2009 SCC, COWS, CASPIS

Collective adaptive systems progr.
- large number of components
- decentralised control
- unpredictable environment
- emergent behaviour

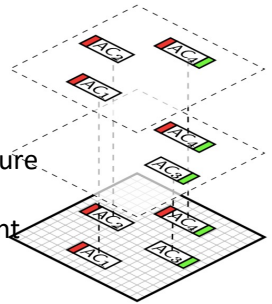
2016 AbC

1998 Klaim



Network-aware programming
- awareness of the network infrastructure
- asynchronous interactions
- open-ended non-determ. environment
- computation mobility

2014 SCEL



Autonomic computing
- reduced maintenance cost
- no human intervention
- continuous monitoring
- adaptation

2020 LAbS

Agent based models
- high-level primitives
- indirect/stigmergic interaction

In this talk

Here, we shall consider:

- AbC: A calculus for Attribute based Programming
  - Modelling Collective Adaptive Systems
- LAbS: A Language with Attribute-based Stigmergies
  - Engineering Agent Based Systems

AbC: A calculus distilled from SCEL

- Components are equipped with attributes whose values can be modified by internal actions and are exposed in their interface .
- Communication actions (send and receive) are decorated with predicates over attributes that partners have to satisfy to make the interaction possible.
- Communication takes place in an implicit multicast fashion: partners are selected via predicates over the attributes exposed in the interfaces.
- Components can offer different views of themselves and can communicate with different partners according to different criteria.
- Semantics for output actions is non-blocking while input actions are blocking and can take place through synchronization with an available sent message.

AbC: Primitives

- Attribute updates $[a := e, \dots]$
  - (Optionally) change values of own attributes
  - Atomically happen after an input/output interaction

- Attribute-based output $(\vec{e})@\Pi[a := e, \dots]$
  - Evaluate expressions $\vec{e}$
  - Send values to all components matching Π

- Attribute-based input $\Pi(\vec{x})[a := e, \dots]$
  - Receive values from any component matching Π
  - Bind values to variables $\vec{x}$

LAbS in few lines

- AbC does not rely on explicit peers identifiers
- Can we also do without explicit send actions?
- In LAbS, agents use stigmergic interaction, i.e. they "drop" bits of information that can be retrieved by others.
- Inspired by biological systems where this form of communication is commonplace
- How?
  - Distributed data structures (*virtual stigmergies*)
  - Shared variables (*environment*)
  - Local information (*attributes*)

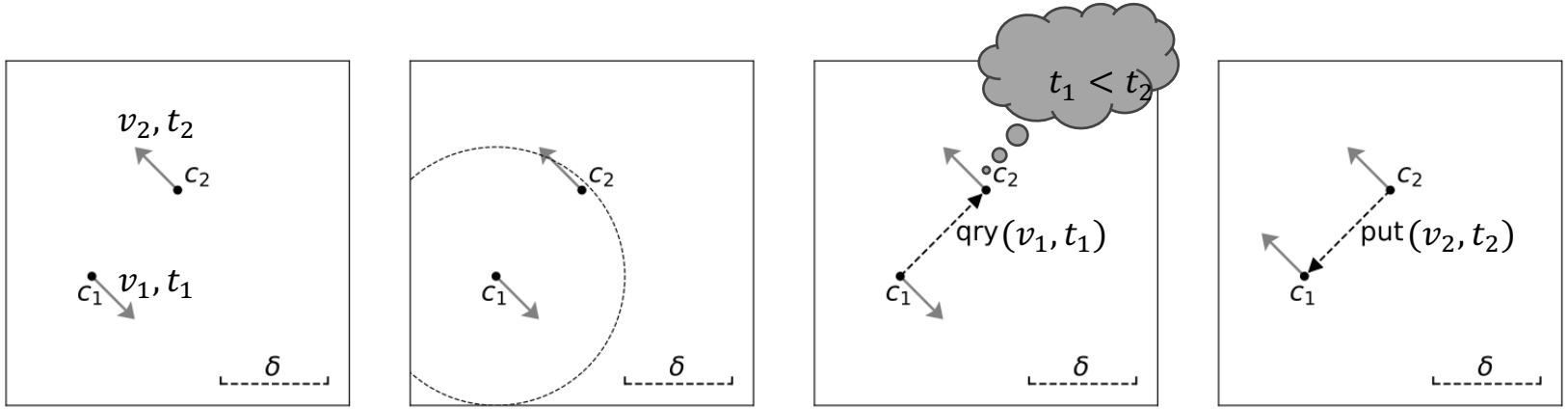
LAbS: Syntax overview

- $P, Q \ ::= \alpha \mid \{B\} \mid P; Q \mid P + Q \mid g \rightarrow P \mid K$ (Process - Componens)
- $\alpha ::= v \leftarrow q \mid v \rightsquigarrow q \mid v \Leftarrow q$ (Statement)
- $\beta ::= \alpha \mid t \ := q \mid p \ := \mathbf{pick}\ n \langle type \rangle \langle \mathbf{where}\ \varphi \rangle$ (Block Statement)
- $B \ ::= \beta \mid \beta\ ; B$ (Block)
- $e \ ::= \kappa \mid v_e \mid e \diamond e \mid \neg e \mid e \wedge e \mid \ e\ \mathbf{if}\ e\ \mathbf{else}\ e$ (Expression)
- $q \ ::= e \mid \mathbf{forall}\ type\ x, q(x) \mid \mathbf{exists}\ type\ x, q(x)$ (Quant. Expression)

- This is the basic syntax of LAbS agents
- Additional rules are needed to describe systems, declare variables, …
- $\leftarrow, \rightsquigarrow, \Leftarrow$ are used to update information about attributes, stigmergy, and environment, respectively.

Virtual stigmergy

- A virtual stigmergy is a store of variables
- At assignment time, a stigmergy variable receives a timestamp
- Agents exchange messages about local values and timestamps after write and read actions; newer values replace older ones
- Messages' exchange is constrained by predicates



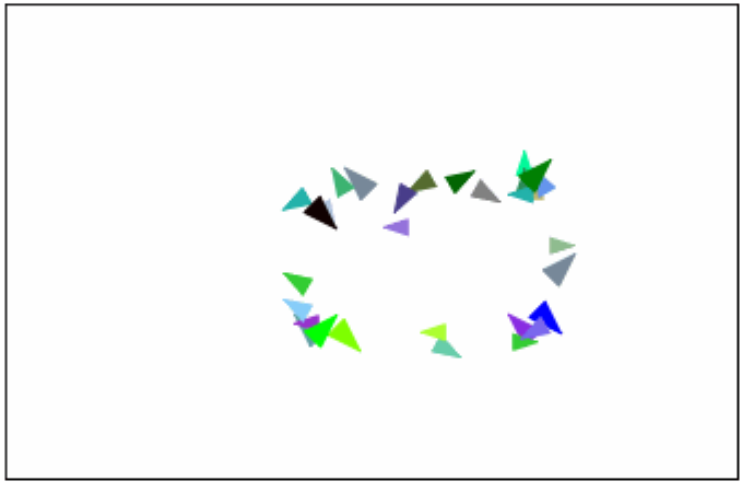
Modelling flocks our way

- High-level language to specify individual behaviours
- Bottom-up, incremental modelling
- Automated encoding into sequential imperative programs
  - For properties verification to exploit existing tools offering different techniques
  - For simulation

Baseline model of a flock of birds

Every agent goes its own way

```
1  agent Bird {
2      Interface =
3          x: 0..G;
4          y: 0..G;
5          dirx: −D..D + 1;
6          diry: −D..D + 1
7
8      Behaviour = Move; Behaviour
9      Move = {
10         x ← x + dirx;
11         y ← y + diry
12     }
13 }
```



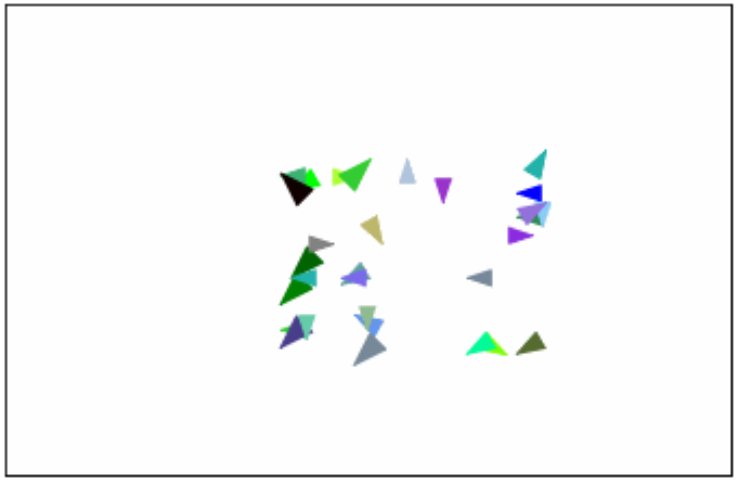
- Attributes of a Bird:
  position (x,y) heading vector (dirx, diry)
- {…} = notation for an atomic block

- $A..B$ = nondeterministic initialization from $[A, B[$
- G, D = external parameters

Alignment: taking the same direction

- Look at another agent and follow its direction

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



pick K returns K agent's identifiers excluding the one of the acting agent

v_p = value of v for agent p

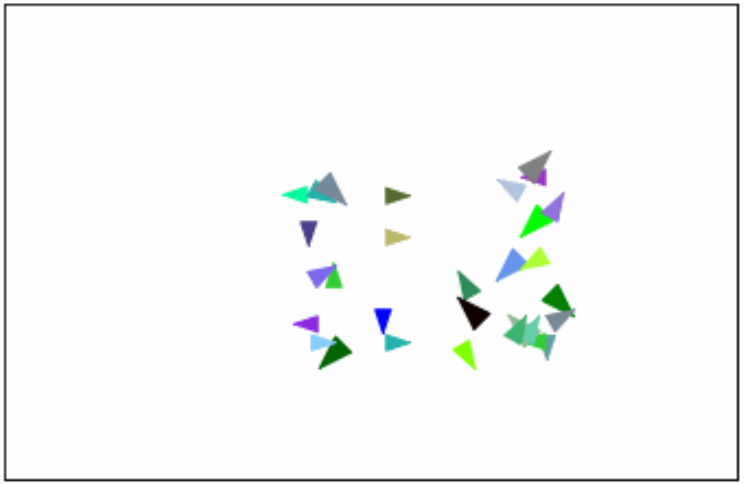
Cohesion: getting as close as possible

Estimate p's position ω steps in the future and aim there; average with current heading for inertia

Move = {
    p := **pick** 1;

    a_x := $x_p + \omega \cdot dirx_p$;
    a_y := $y_p + \omega \cdot diry_p$;
    sgn_x := 0 **if** x = a_x **else** (-1 **if** x > a_x **else** 1);
    diff_x := $d((x, 0), (a\_x, 0))$;
    ... *(Same for* sgn_y, diff_y*)*
    a_dirx := sgn_x $\cdot$ (D:2 **if** diff_y > diff_x **else** D);
    a_diry := sgn_y $\cdot$ (D:2 **if** diff_y < diff_x **else** D);

    dirx $\leftarrow$ (dirx + a_dirx) : 2;
    x $\leftarrow$ x + dirx
    ... *(Same for* diry, y*)*
}



- a **if** c **else** b is the ternary conditional operator
- $x : y$ is real division plus rounding
- $d(a, b)$ is the Manhattan distance $(|x_a - x_b| + |y_a - y_b|)$

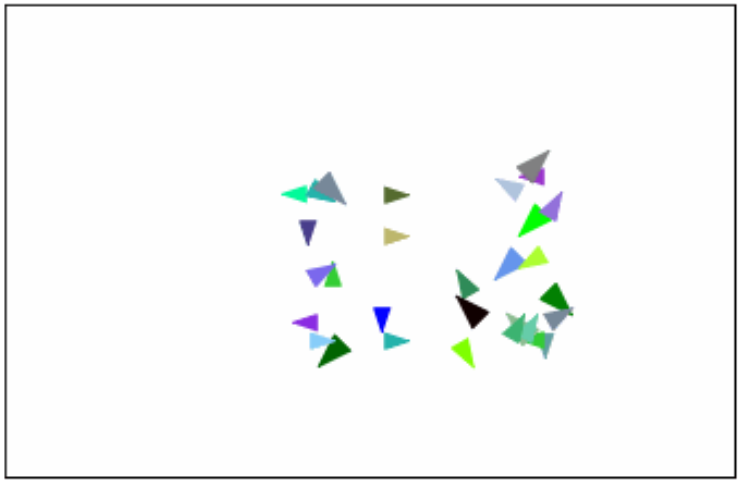
Additional constraints

- Ignore p if it is isolated
- Do not move if destination is already occupied by another bird

Move $= \{$
  p $\coloneqq$ **pick** 1;
  pIsIsolated $\coloneqq$ **forall** Bird b, (b $=$ p) **or** $d((x_p, y_p), (x_b, y_b)) > \delta$;
  appId $\coloneqq$ id **if** pIsIsolated **else** p;

  a_x $\coloneqq x_{appId} + \omega \cdot dirx_{appId}$;
  sgn_x $\coloneqq$ 0 **if** x $=$ a_x **else** (-1 **if** x $>$ a_x, **else** 1);
  diff_x $\coloneqq d((x, 0), (a\_x, 0))$;
  ... *(Same for* a_y, sgn_y, diff_y*)*
  a_dirx $\coloneqq$ sgn_x $\cdot$ (D:2 **if** diff_y $>$ diff_x **else** D);
  a_diry $\coloneqq$ sgn_y $\cdot$ (D:2 **if** diff_y $<$ diff_x **else** D);

  dirx $\leftarrow$ (dirx + a_dirx) : 2;
  diry $\leftarrow$ (diry + a_diry) : 2;
  posFree $\coloneqq$ **forall** Bird b, ($x_b \neq x +$ dirx) **or** ($y_b \neq y +$ diry);
  x $\leftarrow$ x + dirx **if** posFree **else** x
  y $\leftarrow$ y + diry **if** posFree **else** y
$\}$



Decision are taken by evaluating predicates over agents' state

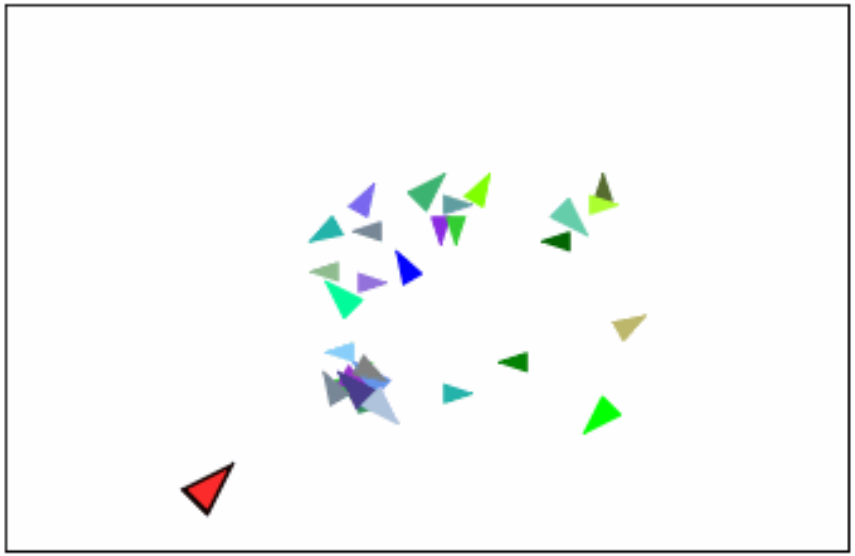
Avoiding predators

- A predator P moving in a straight line is added
- If P is closer than λ, agents move away from it

```
Move = {
    p := pick 1 Bird;
    ...
    a_diry := sgn_y · (D:2 if diff_y < diff_x else D);

    e := pick 1 Predator;
    e_x := x_e + ν · dirx_e;
    esgn_x := 1 if x ≥ e_x else −1;
    ediff_x := d((x, 0), (e_x, 0));
    ... (Same for e_y, esgn_y, ediff_y)
    e_dirx := esgn_x · (D:2 if ediff_y > ediff_x else D);
    e_diry := esgn_y · (D:2 if ediff_y < ediff_x else D);

    e_dist := d((x, y), (e_x, e_y));
    f_dirx := e_dirx if e_dist < λ else a_dirx;
    dirx ← (dirx + f_dirx) : 2;
    ... (Same for f_diry, diry)
    posFree := forall Bird b, (x_b ≠ x + dirx) or (y_b ≠ y + diry);
    x ← x + dirx if posFree else x
}
```



This time **pick** is typed

Concerns about global information

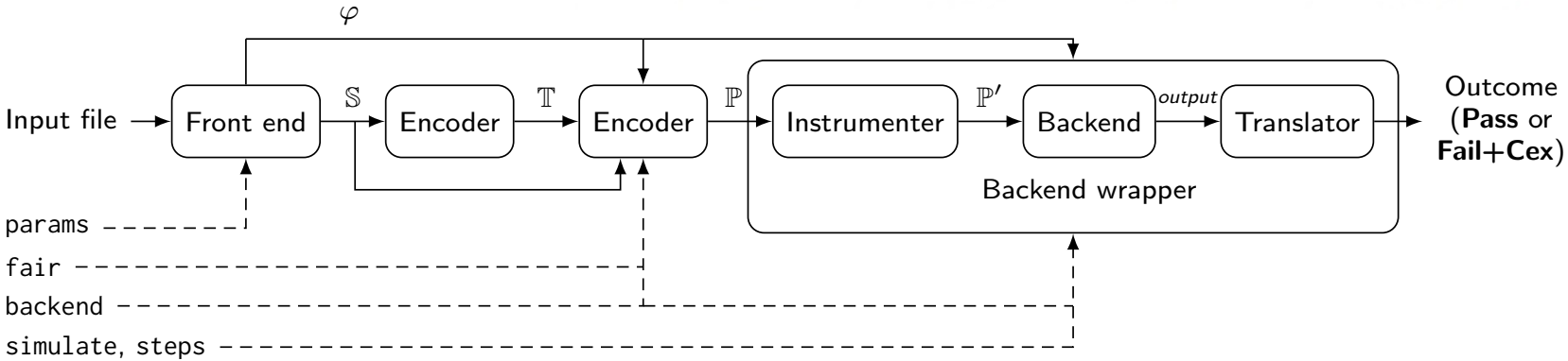
- Formally, pick, forall, exists, v_p rely on a global knowledge of the system

- However, they can be evaluated "locally" by a given agent
  - "pick 1 agent randomly" -→ a form of neighbor discovery
  - "check that no agent is at position (x, y)" -→ look at (x, y)
  - $dirx_p$ -→ look at agent p
  - "check that agent p is isolated" -→ look around agent p

Additional details

- For this model we assumed round-robin scheduling, allowing the execution of an instruction (or an atomic block) by each agent at each round
- We can set up initial constraints in an **assume** { … } section, e.g., to state that:
  - Birds start at the center of the arena
  - Birds have different initial positions
  - Birds have non-null initial directions
- For simulation, some nondeterminism is resolved upfront by fixing initial states and outcomes of **pick** statements

SLiVER: Verification/Simulation of LAbS models



- 𝕊 Syntax tree of model
- 𝕋 Intermediate representation (a set of triples: activation condition, statement, program counter update)
- ℙ Emulation program (here a C program; we also support LNT)

The back end currently used is CBMC (SAT-based bounded model checking)

Verifying cohesion

In the model without the predator, we want to check that the distance between all birds gets smaller than k after B steps:

$$\textbf{after } B \textbf{ forall } \text{Bird } a, \textbf{forall } \text{Bird } b, d(a,b) < k$$

Indeed: If initially all birds are "close" they keep close.

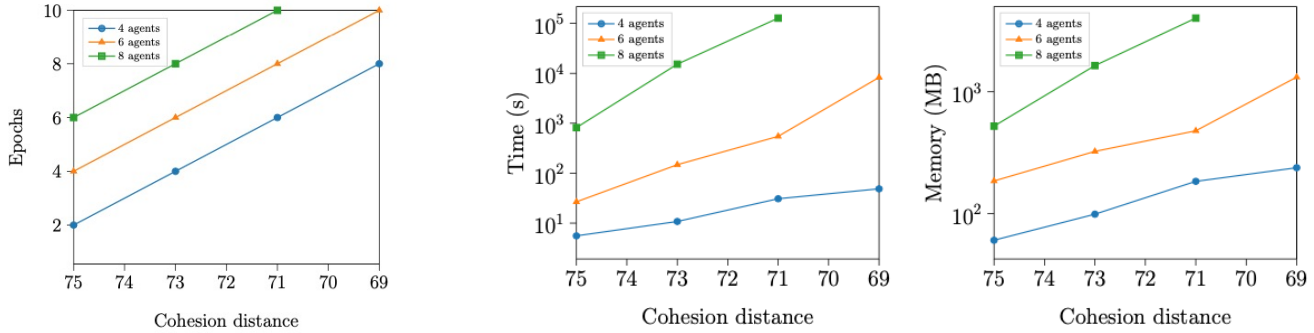
But: With two groups at distance $k_0 > k$, the property may not hold.

- **pick** might always select bird from the same group.
- To avoid this, we assign Birds a $groupId$ and force them to look at a different group after each step (**pick** … **where** … adds further constraints when choosing agents)

$$p \coloneqq \textbf{pick}\ 1\ \textbf{where}\ (check = 0)\ \textbf{or}\ \left(groupId_p \neq groupId\right);$$

$$check = (check + 1)\textbf{mod}\ 2;$$

- With this fix the property holds for several values of k, B



Conclusion

- Bottom-up modelling with LAbS
  - Compact specifications of individual behaviours
  - No explicit communication primitives
  - Possibility of gradual improvements
- Simulations
  - Some assurance that the desired collective behaviour will emerge
  - Feedback to refine specifications
- Verification
  - Catch subtle bugs
  - Can definitely prove the emergence of desired collective features

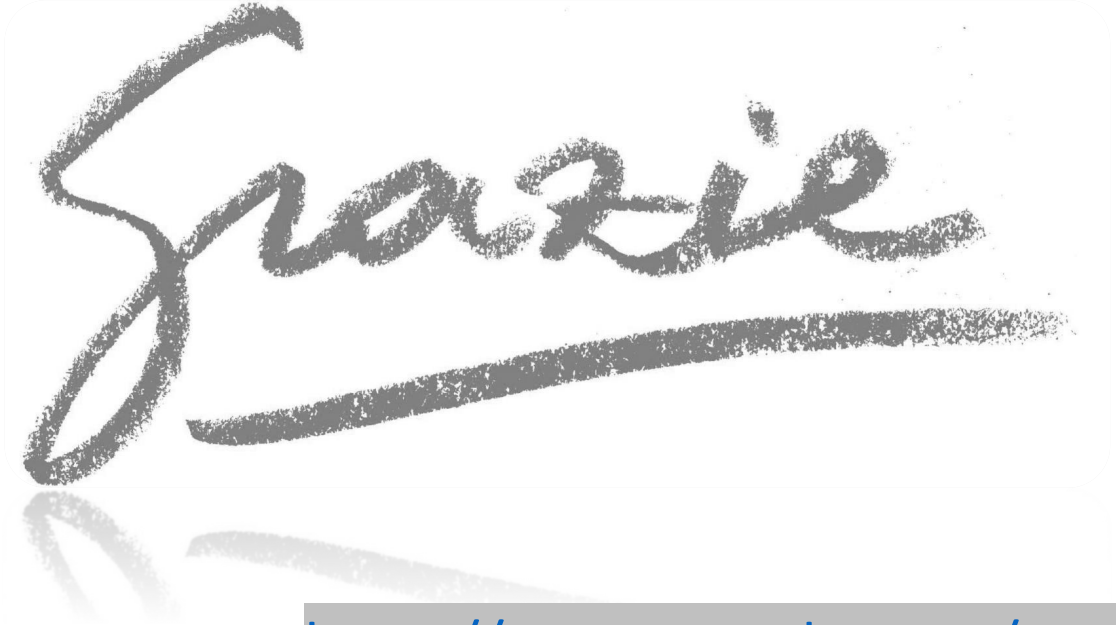
Next steps

- Modelling flocks in 3D
- Statistical model checking
- Runtime verification
- Use other verification back ends
- Extend the range of supported properties
- Consider different approaches and compare
  - Discrete-time dynamics (e.g., Boids)
  - Equational modelling (
  - Decentralized control laws

References

- L. Di Stefano, R. De Nicola, O. Inverso: Verification of Distributed Systems via Sequential Emulation. ACM Trans. Softw. Eng. Methodol. 31(3): 37:1-37:41 (2022)
- R. De Nicola, T. Duong, M. Loreti: Provably correct implementation of the AbC calculus. Sci. Comput. Program. 202: 102567, Elsevier 2021.
- R. De Nicola, G.L. Ferrari, R. Pugliese, F. Tiezzi: A formal approach to the engineering of domain- specific distributed systems. J. Log. Algebraic Methods Program. 111: 100511, Elsevier 2020.
- Y. Abd Alrahman, R. De Nicola, M. Loreti: Programming interactions in collective adaptive systems by relying on attribute-based communication. Sci. Comput. Program. 192: 102428, Elsevier 2020.
- R. De Nicola, L. Di Stefano, O. Inverso: Multi-agent systems with virtual stigmergy. Sci. Comput. Program., 187, Elsevier 2020.
- R. De Nicola, L. Di Stefano, O. Inverso, S. Valiani: Modelling Flocks of Birds from the Bottom Up, in ISoLA, LNCS 13703, Springer 2022
- R. De Nicola, L. Di Stefano, O. Inverso, S. Valiani: Process algebras and flocks of birds, in A journey from process algebra via timed automata to model learning, LNCS 13560, Springer 2022.
- Di Stefano's Thesis: https://hdl.handle.net/20.500.12571/10181
- LAbS code: https://github.com/labs-lang

QUESTIONS?



https://www.youtube.com/watch?v=3w90X92pDSs