Formal Proofs for Mobile Robot Swarms

Lionel Rieg

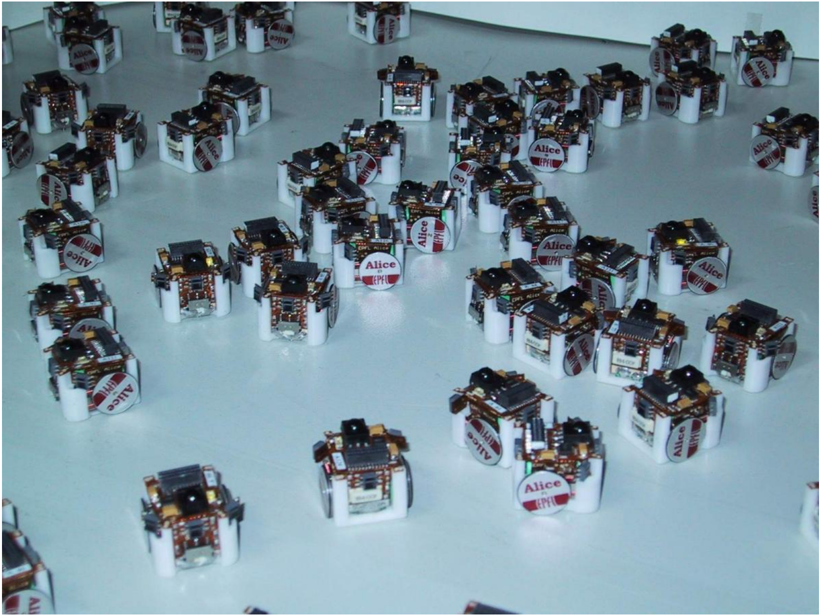
Ensimag – Grenoble INP / VERIMAG

Réunion Estate/Descartes, Roscoff, 2 avril 2019

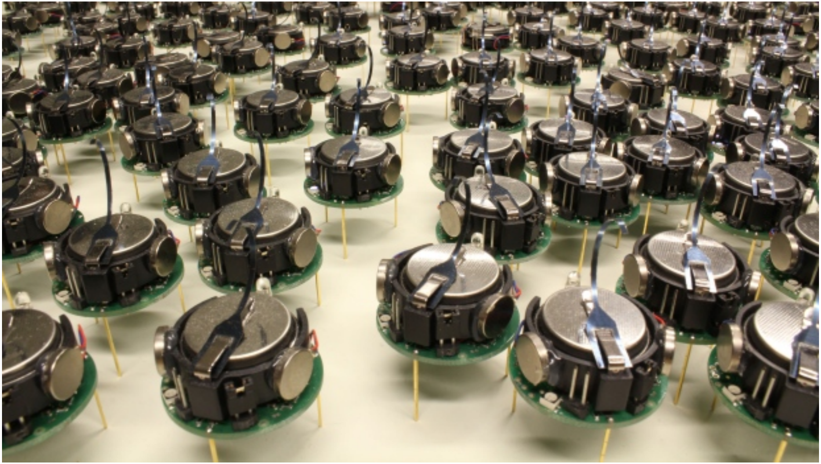
Inspiration: Swarms of Mobile Robots

- Swarms?

Inspiration: Swarms of Mobile Robots



Inspiration: Swarms of Mobile Robots



Inspiration: Swarms of Mobile Robots

- Swarms?
  - lots of (small) identical robots

Inspiration: Swarms of Mobile Robots

- Swarms?
  - lots of (small) identical robots

- Where?

Inspiration: Swarms of Mobile Robots

Inspiration: Swarms of Mobile Robots

Inspiration: Swarms of Mobile Robots

- Swarms?
  - lots of (small) identical robots
- Where?
  - entertainment
  - rescue
  - exploration
  - ...

Inspiration: Swarms of Mobile Robots

- Swarms?
  - lots of (small) identical robots
- Where?
  - entertainment
  - rescue
  - exploration
  - ...
- Opportunities?
  - cooperative behavior (swarm intelligence)
  - resilience
- Main challenge?
  - Understand what happens!

Why Formal Methods for Mobile Robots Swarms?

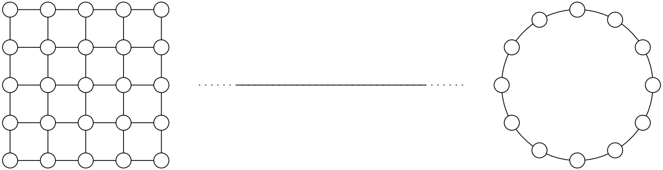
Many mobile robots swarm models:

- Space — discrete/continuous, bounded/unbounded, topology, ...
- Sensors — multiplicity, range, accuracy, orientation, ...
- Faults — none, crash, Byzantine, ...
- Execution — synchronous/asynchronous, fairness, interruption, ...

Why Formal Methods for Mobile Robots Swarms?

Many mobile robots swarm models:

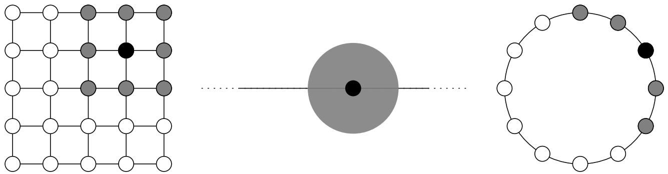
- Space discrete/continuous, bounded/unbounded, topology, . . .
- Sensors multiplicity, range, accuracy, orientation, . . .
- Faults none, crash, Byzantine, . . .
- Execution synchronous/asynchronous, fairness, interruption, . . .



Why Formal Methods for Mobile Robots Swarms?

Many mobile robots swarm models:

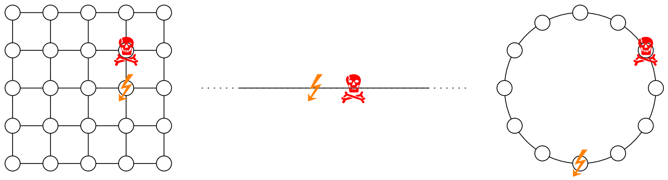
- Space discrete/continuous, bounded/unbounded, topology, ...
- Sensors multiplicity, range, accuracy, orientation, ...
- Faults none, crash, Byzantine, ...
- Execution synchronous/asynchronous, fairness, interruption, ...



Why Formal Methods for Mobile Robots Swarms?

Many mobile robots swarm models:

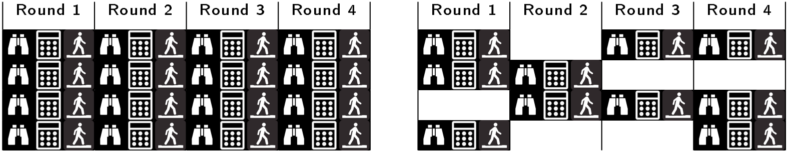
- Space — discrete/continuous, bounded/unbounded, topology, ...
- Sensors — multiplicity, range, accuracy, orientation, ...
- Faults — none, crash, Byzantine, ...
- Execution — synchronous/asynchronous, fairness, interruption, ...

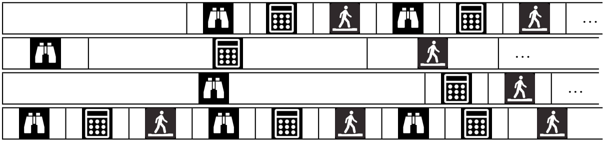


Why Formal Methods for Mobile Robots Swarms?

Many mobile robots swarm models:

- Space — discrete/continuous, bounded/unbounded, topology, ...
- Sensors — multiplicity, range, accuracy, orientation, ...
- Faults — none, crash, Byzantine, ...
- Execution — synchronous/asynchronous, fairness, interruption, ...





Why Formal Methods for Mobile Robots Swarms?

Many mobile robots swarm models:

- Space discrete/continuous, bounded/unbounded, topology, ...
- Sensors multiplicity, range, accuracy, orientation, ...
- Faults none, crash, Byzantine, ...
- Execution synchronous/asynchronous, fairness, interruption, ...

- Subtle differences in models $\rightsquigarrow$ Very error-prone
- Careful of mismatch spec/proof!
- Lots of proof cases

$\Rightarrow$ Formal methods can help

Why Formal Methods for Mobile Robots Swarms?

Many mobile robots swarm models:

- Space — discrete/continuous, bounded/unbounded, topology, ...
- Sensors — multiplicity, range, accuracy, orientation, ...
- Faults — none, crash, Byzantine, ...
- Execution — synchronous/asynchronous, fairness, interruption, ...

- Subtle differences in models $\rightsquigarrow$ Very error-prone
- Careful of mismatch spec/proof!
- Lots of proof cases

$\Rightarrow$ Formal methods can help

Which model? (process algebra, TLA, ...)
Which tool? (model checking, proof assistant, ...)

We need a suitable framework — What are the requirements?

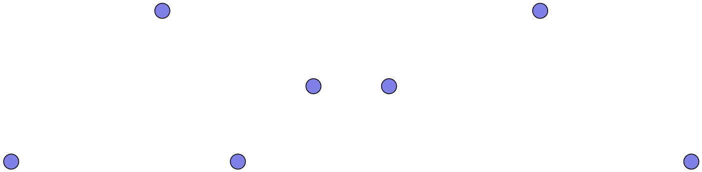
Robot Model

[Suzuki, Yamashita 99]

Robot Model

[Suzuki, Yamashita 99]

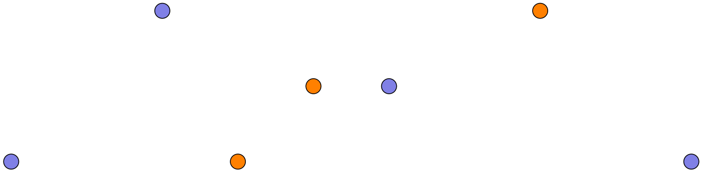
- Points



Robot Model

[Suzuki, Yamashita 99]

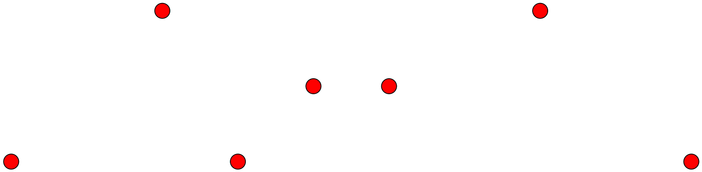
- Points
- With Byzantine faults (or crash)



Robot Model

[Suzuki, Yamashita 99]

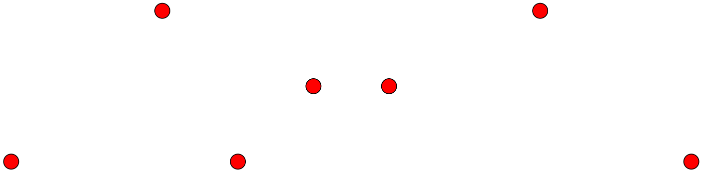
- Points
- With Byzantine faults (or crash)
- Anonymous



Robot Model

[Suzuki, Yamashita 99]

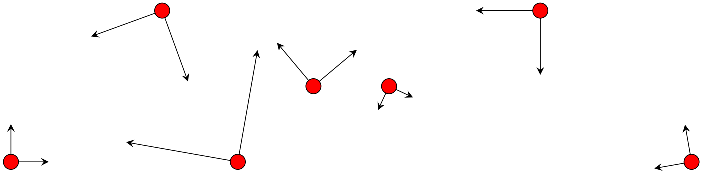
- Points
- With Byzantine faults (or crash)
- Anonymous
- No direct communication



Robot Model

[Suzuki, Yamashita 99]

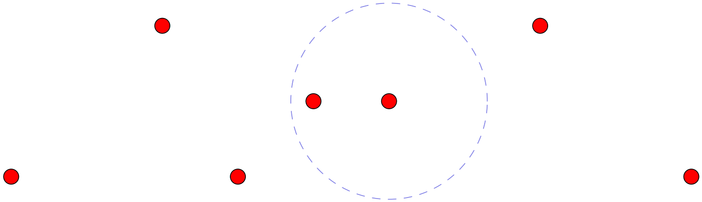
- Points
- With Byzantine faults (or crash)
- Anonymous
- No direct communication
- No common frame/direction
  $\rightsquigarrow$ local coordinates



Robot Model

[Suzuki, Yamashita 99]

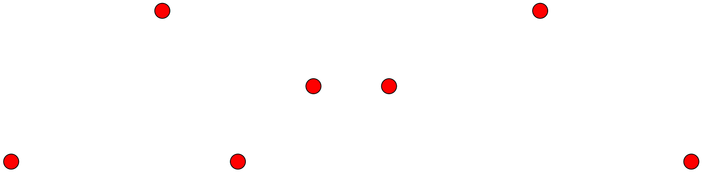
- Points
- With Byzantine faults (or crash)
- Anonymous
- No direct communication
- No common frame/direction
  $\rightsquigarrow$ local coordinates
- Limited/unlimited vision? multiplicity?



Robot Model

[Suzuki, Yamashita 99]

- Points
- With Byzantine faults (or crash)
- Anonymous
- No direct communication
- No common frame/direction
  $\rightsquigarrow$ local coordinates
- Limited/unlimited vision? multiplicity?
- Same (deterministic) program everywhere



Execution Models (1)

[Suzuki, Yamashita 99]

3 phases for each robot:

1. Look: observe its surrounding
   - indirect communication
   - depends on sensor capabilities
2. Compute: choose what to do
   - choose an objective
   - depends on observation, program
3. Move: do it (or try to)
   - try to reach your target
   - depends on the environment

Execution Models (1)

[Suzuki, Yamashita 99]

3 phases for each robot:

1. Look: observe its surrounding
   - indirect communication
   - depends on sensor capabilities
2. Compute: choose what to do
   - choose an objective
   - depends on observation, program
3. Move: do it (or try to)
   - try to reach your target
   - depends on the environment

and repeat

Execution Models (2): Scheduling [Suzuki, Yamashita 99]

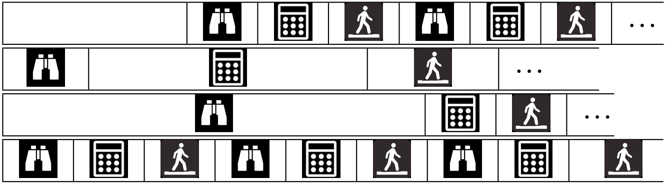
Scheduling of robots is

Execution Models (2): Scheduling [Suzuki, Yamashita 99]

Scheduling of robots is

Either ASYNC: full interleaving

- Most general/realistic but hardest



Execution Models (2): Scheduling

[Suzuki, Yamashita 99]

Scheduling of robots is

Either ASYNC: full interleaving

- Most general/realistic but hardest

or Same phase for all active robots

- Time split into rounds

Round 1	Round 2	Round 3	Round 4	Round 5

Execution Models (2): Scheduling

[Suzuki, Yamashita 99]

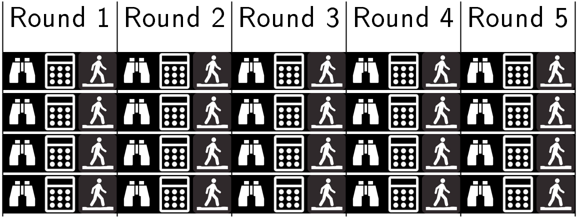
Scheduling of robots is

Either ASYNC: full interleaving

- Most general/realistic but hardest

or Same phase for all active robots

- Time split into rounds
- FSYNC: all robots are activated each round



Execution Models (2): Scheduling [Suzuki, Yamashita 99]

Scheduling of robots is

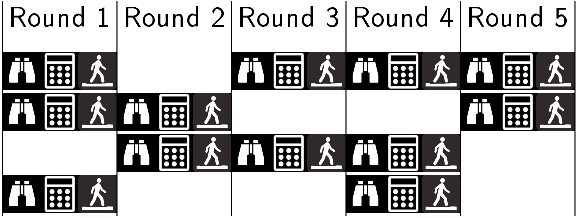
Either ASYNC: full interleaving

- Most general/realistic but hardest

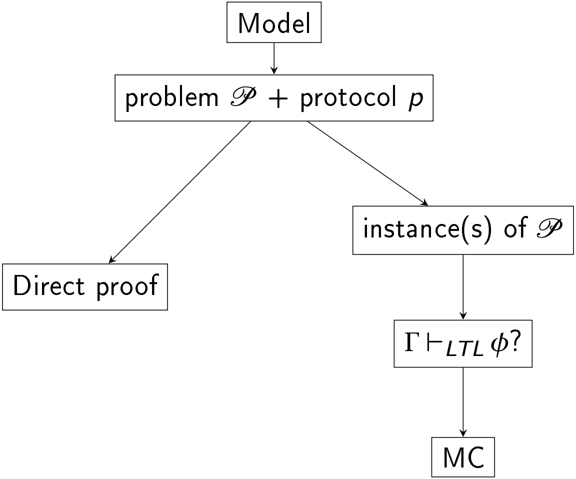
or Same phase for all active robots

- Time split into rounds
- FSYNC: all robots are activated each round
- SSYNC: only a subset is activated

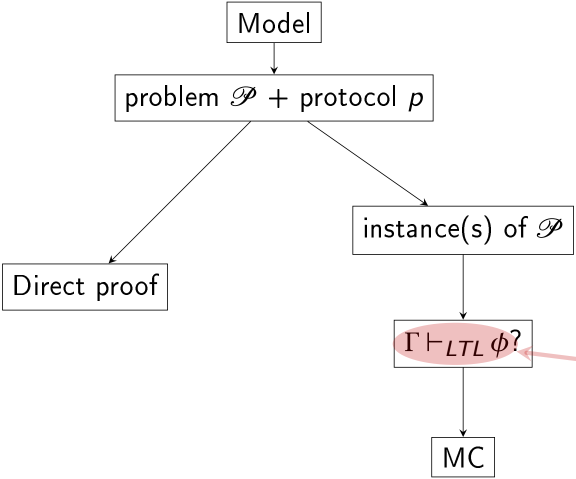
  $\rightsquigarrow$ Fairness assumptions on the scheduling (demon)



Proving Distributed Protocol

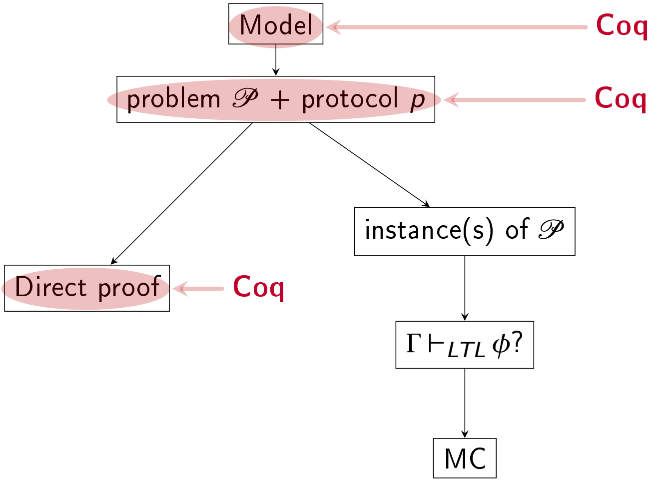


Proving Distributed Protocol



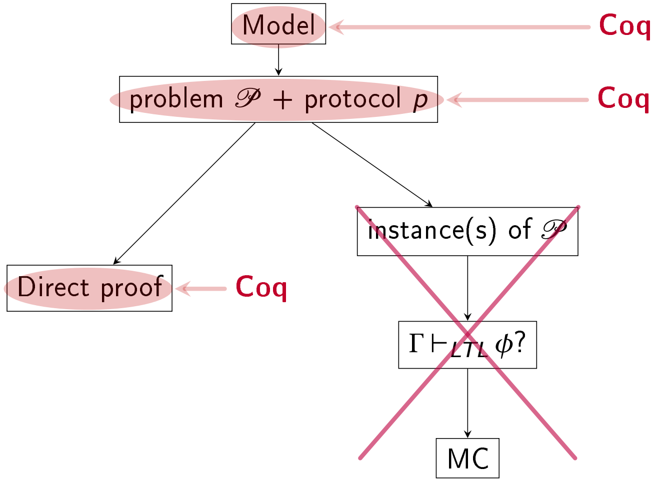
true iff p succeeds on all schedulings

Pactole



Model
Coq
problem $\mathscr{P}$ + protocol p
Coq
instance(s) of $\mathscr{P}$
Direct proof
Coq
$\Gamma \vdash_{LTL} \phi$?
MC

Pactole



Model
Coq
problem $\mathscr{P}$ + protocol p
Coq
instance(s) of $\mathscr{P}$
Direct proof
Coq
$\Gamma \vdash_{LTL} \phi$?
MC

Pactole: a Coq Framework for Mobile Robots

Very Parametric (but still useful):

- Space
- State of Robots (location, memory, battery level, etc.)
- Sensors
- How states are updated during the move phase

Key Ingredients:

- Configuration — function ident → state
- Spectrum — info on config from sensors
- Robogram — function spectrum → location
- Demon (scheduler) — adversarial environment
- Round — one step of execution

Pactole: a Coq Framework for Mobile Robots

Very Parametric (but still useful):

- Space
- State of Robots (location, memory, battery level, etc.)
- Sensors
- How states are updated during the move phase

Key Ingredients:

- Configuration — function ident → state
- Spectrum — info on config from sensors
- Robogram — function spectrum → location
- Demon (scheduler) — adversarial environment
- Round — one step of execution

Description of a Round

Robogram: what happens for a robot?

Demon: what does the environment decide?

Description of a Round

Robogram: what happens for a robot?

1. If it is not activated, some update happens (for ASYNC only)

Demon: what does the environment decide?

Description of a Round

Robogram: what happens for a robot?

1. If it is not activated, some update happens (for ASYNC only)

Demon: what does the environment decide?

1. Pick which robots are activated
2. Decide how to update inactive ones

Description of a Round

Robogram: what happens for a robot?

1. If it is not activated, some update happens (for ASYNC only)
2. If it is activated and Byzantine, the demon gives its new state

Demon: what does the environment decide?

1. Pick which robots are activated
2. Decide how to update inactive ones

Description of a Round

Robogram: what happens for a robot?

1. If it is not activated, some update happens (for ASYNC only)
2. If it is activated and Byzantine, the demon gives its new state

Demon: what does the environment decide?

1. Pick which robots are activated
2. Decide how to update inactive ones
3. Update Byzantine ones as it wishes

Description of a Round

Robogram: what happens for a robot?

1. If it is not activated, some update happens (for ASYNC only)
2. If it is activated and Byzantine, the demon gives its new state
3. If it is activated and not Byzantine (i.e. Good),
   a. **Look**: get information from its surrounding
   b. **Compute** its destination
   c. **Move** to the destination

Demon: what does the environment decide?

1. Pick which robots are activated
2. Decide how to update inactive ones
3. Update Byzantine ones as it wishes

Description of a Round

Robogram: what happens for a robot?

1. If it is not activated, some update happens (for ASYNC only)
2. If it is activated and Byzantine, the demon gives its new state
3. If it is activated and not Byzantine (i.e. Good),
   a. **Look**: get information from its surrounding
   b. **Compute** its destination
   c. **Move** to the destination

Demon: what does the environment decide?

1. Pick which robots are activated
2. Decide how to update inactive ones
3. Update Byzantine ones as it wishes
4. Select the new frame of reference for non Byzantine ones
5. Decide how to update them depending on their destination

The Core of Pactole: the Round Function

Coq

```
Definition round (r:robogram) (da:demonic_action) cfg
 : configuration :=
 fun id ⇒    (* for a given robot, we compute the new state *)
```

The Core of Pactole: the Round Function

Coq

```
Definition round (r:robogram) (da:demonic_action) cfg
 : configuration :=
 fun id ⇒    (* for a given robot, we compute the new state *)
  if da.(activate) id (* see whether the robot is activated *)
  then



















  else inactive cfg id (da.(choose_inactive) cfg id).
```

The Core of Pactole: the Round Function

Coq

```
Definition round (r:robogram) (da:demonic_action) cfg
 : configuration :=
 fun id ⇒    (* for a given robot, we compute the new state *)
  if da.(activate) id (* see whether the robot is activated *)
  then
   match id with
   | Byz b ⇒ da.(relocate_byz) cfg b              (* byzantine *)
   | Good g ⇒
















   end
  else inactive cfg id (da.(choose_inactive) cfg id).
```

The Core of Pactole: the Round Function

Coq

```
Definition round (r:robogram) (da:demonic_action) cfg
 : configuration :=
 fun id ⇒    (* for a given robot, we compute the new state *)
  if da.(activate) id (* see whether the robot is activated *)
  then
   match id with
   | Byz b ⇒ da.(relocate_byz) cfg b            (* byzantine *)
   | Good g ⇒
     (* change the frame of reference *)
     let new_frame := frame_choice (da.(change_frame) cfg g) in
     let local_cfg := map_cfg (lift new_frame ( ... )) cfg in
     let local_pos := get_location (local_cfg (Good g)) in









   end
  else inactive cfg id (da.(choose_inactive) cfg id).
```

The Core of Pactole: the Round Function

Coq

```
Definition round (r:robogram) (da:demonic_action) cfg
 : configuration :=
 fun id ⇒    (* for a given robot, we compute the new state *)
  if da.(activate) id (* see whether the robot is activated *)
  then
   match id with
   | Byz b ⇒ da.(relocate_byz) cfg b          (* byzantine *)
   | Good g ⇒
     (* change the frame of reference *)
     let new_frame := frame_choice (da.(change_frame) cfg g) in
     let local_cfg := map_cfg (lift new_frame ( ... )) cfg in
     let local_pos := get_location (local_cfg (Good g)) in
     (* compute the spectrum *)
     let spect := spect_from_cfg local_cfg local_pos in









   end
  else inactive cfg id (da.(choose_inactive) cfg id).
```

The Core of Pactole: the Round Function

Coq

```
Definition round (r:robogram) (da:demonic_action) cfg
 : configuration :=
 fun id ⇒    (* for a given robot, we compute the new state *)
  if da.(activate) id (* see whether the robot is activated *)
  then
   match id with
   | Byz b ⇒ da.(relocate_byz) cfg b              (* byzantine *)
   | Good g ⇒
     (* change the frame of reference *)
     let new_frame := frame_choice (da.(change_frame) cfg g) in
     let local_cfg := map_cfg (lift new_frame ( ... )) cfg in
     let local_pos := get_location (local_cfg (Good g)) in
     (* compute the spectrum *)
     let spect := spect_from_cfg local_cfg local_pos in
     (* apply r on spectrum *)
     let local_traj := r spect in







   end
  else inactive cfg id (da.(choose_inactive) cfg id).
```

The Core of Pactole: the Round Function

Coq

```
Definition round (r:robogram) (da:demonic_action) cfg
 : configuration :=
 fun id ⇒    (* for a given robot, we compute the new state *)
  if da.(activate) id (* see whether the robot is activated *)
  then
   match id with
   | Byz b ⇒ da.(relocate_byz) cfg b              (* byzantine *)
   | Good g ⇒
     (* change the frame of reference *)
     let new_frame := frame_choice (da.(change_frame) cfg g) in
     let local_cfg := map_cfg (lift new_frame ( ... )) cfg in
     let local_pos := get_location (local_cfg (Good g)) in
     (* compute the spectrum *)
     let spect := spect_from_cfg local_cfg local_pos in
     (* apply r on spectrum *)
     let local_traj := r spect in
     (* return to the global frame of reference *)
     let global_traj := lift_path (new_frame ⁻¹) local_traj in




   end
  else inactive cfg id (da.(choose_inactive) cfg id).
```

The Core of Pactole: the Round Function

Coq

```
Definition round (r:robogram) (da:demonic_action) cfg
 : configuration :=
 fun id ⇒    (* for a given robot, we compute the new state *)
  if da.(activate) id (* see whether the robot is activated *)
  then
   match id with
   | Byz b ⇒ da.(relocate_byz) cfg b              (* byzantine *)
   | Good g ⇒
     (* change the frame of reference *)
     let new_frame := frame_choice (da.(change_frame) cfg g) in
     let local_cfg := map_cfg (lift new_frame ( ... )) cfg in
     let local_pos := get_location (local_cfg (Good g)) in
     (* compute the spectrum *)
     let spect := spect_from_cfg local_cfg local_pos in
     (* apply r on spectrum *)
     let local_traj := r spect in
     (* return to the global frame of reference *)
     let global_traj := lift_path (new_frame ⁻¹) local_traj in
     (* the demon chooses how to perform the state update *)
     let choice := da.(choose_update) cfg g global_traj in


   end
  else inactive cfg id (da.(choose_inactive) cfg id).
```

The Core of Pactole: the Round Function

Coq

```
Definition round (r:robogram) (da:demonic_action) cfg
 : configuration :=
 fun id ⇒    (* for a given robot, we compute the new state *)
  if da.(activate) id (* see whether the robot is activated *)
  then
   match id with
    | Byz b ⇒ da.(relocate_byz) cfg b              (* byzantine *)
    | Good g ⇒
      (* change the frame of reference *)
      let new_frame := frame_choice (da.(change_frame) cfg g) in
      let local_cfg := map_cfg (lift new_frame ( ... )) cfg in
      let local_pos := get_location (local_cfg (Good g)) in
      (* compute the spectrum *)
      let spect := spect_from_cfg local_cfg local_pos in
      (* apply r on spectrum *)
      let local_traj := r spect in
      (* return to the global frame of reference *)
      let global_traj := lift_path (new_frame ⁻¹) local_traj in
      (* the demon chooses how to perform the state update *)
      let choice := da.(choose_update) cfg g global_traj in
      (* actual state update performed by the update function *)
      update cfg g global_traj choice
   end
  else inactive cfg id (da.(choose_inactive) cfg id).
```

Proving in Pactole

Correctness proof:

1. Formalize your problem

Proving in Pactole

Correctness proof:

1. Formalize your problem
2. Write your robogram

Proving in Pactole

Correctness proof:

1. Formalize your problem
2. Write your robogram
3. Express your algorithm in the global frame of reference

Two different points of view

Global View (demon + proof)	**Local View** (robots)
absolute location	local frame of reference
robots: Byzantine or not (B / G)	indistinguishable robots
identifiers ident = $G + B$	
configuration = ident → position	local configuration
	spectrum
	robogram
	: spectrum → location
round r d : config → config	
exec = stream of configurations	

spectrum = degraded local view of a configuration

Proving in Pactole

Correctness proof:

1. Formalize your problem
2. Write your robogram
3. Express your algorithm in the global frame of reference

Proving in Pactole

Correctness proof:

1. Formalize your problem
2. Write your robogram
3. Express your algorithm in the global frame of reference
   Use geometric patterns that are invariant by change of frame

```
Lemma round_simplify :∀da cfg, round r da cfg ≡...
```

Proving in Pactole

Correctness proof:

1. Formalize your problem
2. Write your robogram
3. Express your algorithm in the global frame of reference
   Use geometric patterns that are invariant by change of frame
   ```
   Lemma round_simplify :∀da cfg, round r da cfg ≡...
   ```
4. Prove that your algorithm solves your problem
   following your paper proof

Proving in Pactole

Correctness proof:

1. Formalize your problem
2. Write your robogram
3. Express your algorithm in the global frame of reference
   Use geometric patterns that are invariant by change of frame
   ```
   Lemma round_simplify :∀da cfg, round r da cfg ≡...
   ```
4. Prove that your algorithm solves your problem
   following your paper proof

Impossibility proof:

1. Formalize your problem
2. Assume given a robogram (a variable) + its properties
3. Prove that the algorithm does not solve the problem
   (`round_simplify` not always needed)

Some results (✓verified in Pactole)

- Convergence $\mathbb{R}^2$, rigid, $\geq 1/3$ Byzantine, SSYNC
  Impossibility [Bouzid et al. TCS'10] [SSS'13] ✓
- Gathering $\mathbb{R}^2$, rigid, no Byzantine, SSYNC
  Impossibility ($2 \times n$ robots) [Suzuki,Yamashita SICOMP'99] [IPL'15] ✓
- Gathering $\mathbb{R}^2$, rigid, no Byzantine, SSYNC, start not bivalent
  Effective (universal) algorithm [DISC'16] ✓
- Gathering $\mathbb{R}^2$, non rigid, no Byzantine, FSYNC, no multiplicity
  Effective (universal) algorithm [SSS'16] ✓
- Exploration $\mathbb{Z}/n\mathbb{Z}$, discrete, k robots, no bizantine, SSYNC
  Impossibility (k divides n) [ICDCN'17] ✓
- Discrete/continuous graphs, non rigid, ASYNC
  Model equivalence [SSS'18] ✓
- Connection $\mathbb{R}^2$, rigid, no bizantine, SSYNC
  Effective (non optimal) algorithm [SSS'21] ✓

Case Study: Gathering

Objective

Have all (non byzantine) robots reach in finite time the same location (unknown ahead of time) and then stay there.

Let's formalize it!

Case Study: Gathering

Objective

Have all (non byzantine) robots reach in finite time the same location (unknown ahead of time) and then stay there.

Let's formalize it!

Coq

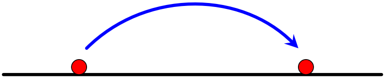
```
(* All good robots are at the same location [pt] (exactly). *)
Definition gathered_at (pt : location) (cfg : configuration) :=
  ∀ g, get_location (cfg (Good g)) ≡pt.

(* At all rounds of [e], robots are gathered at [pt]. *)
Definition Gather (pt : location) (e : execution) : Prop :=
  Stream.forever (Stream.instant (gathered_at pt)) e.

(* The infinite execution [e] is *eventually* [Gather]ed. *)
Definition WillGather (pt : location) (e : execution) : Prop :=
  Stream.eventually (Gather pt) e.
```

Impossibility Proof [Suzuki & Yamashita 99]

By symmetry, both robots act the same.

Two cases:

Impossibility Proof [Suzuki & Yamashita 99]

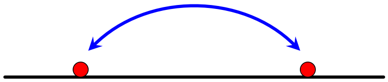


By symmetry, both robots act the same.

Two cases:

1. Left robot moves to the right one

Impossibility Proof [Suzuki & Yamashita 99]



By symmetry, both robots act the same.

Two cases:

1. Left robot moves to the right one
   activate both: swap locations

Impossibility Proof [Suzuki & Yamashita 99]

By symmetry, both robots act the same.

Two cases:

2. Left robot goes anywhere else

Impossibility Proof [Suzuki & Yamashita 99]

By symmetry, both robots act the same.

Two cases:

2. Left robot goes anywhere else
   activate only the left one: same configuration up to scale

Impossibility Proof [Suzuki & Yamashita 99]

By symmetry, both robots act the same.

Two cases:

1. Left robot moves to the right one
   activate both: swap locations
2. Left robot goes anywhere else
   activate only the left one: same configuration up to scale

In both cases, a similar configuration at the next round

Impossibility Proof [Suzuki & Yamashita 99]

By symmetry, both robots act the same.

Two cases:

1. Left robot moves to the right one
   activate both: swap locations
2. Left robot goes anywhere else
   activate only the left one: same configuration up to scale

In both cases, a similar configuration at the next round

Generalizations:

- Even number of robots
- Type of line ($\mathbb{Q}$ ou $\mathbb{R}$), higher dimension

Universal Algorithm

Key Ideas:

- Center of the smallest enclosing circle
- Move in stages (not all robots at once)
- Never create the configuraton of the impossibility proof

Hard part of the proof: termination $\leadsto$ termination order

- #robots on the circle decreases
- Stages
- Avoid looping with triangles

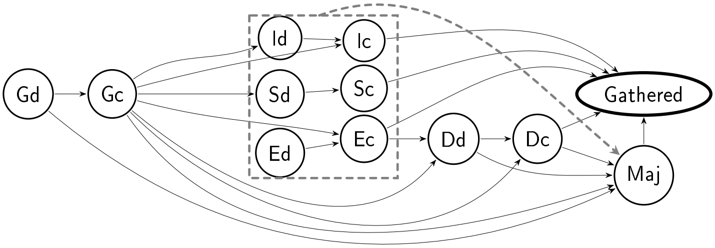
Universal Algorithm

Key Ideas:

- Center of the smallest enclosing circle
- Move in stages (not all robots at once)
- Never create the configuraton of the impossibility proof

Hard part of the proof: termination $\rightsquigarrow$ termination order

- #robots on the circle decreases
- Stages
- Avoid looping with triangles



Termination Order in Coq

Coq

```
Definition m_clean s := nG-s[target s].
Definition m_dirty s := nG-SECT_cardinal s.
Function measure (s : observation): nat * nat :=
 match support (max s) with
  | nil ⇒ (0, 0)                                        (* no robot *)
  | pt :: nil ⇒ (0, nG -  s[pt])                        (* majority *)
  | _ :: _ :: _ ⇒
    match on_SEC (support s) with
     | nil | _::nil ⇒ (0,0)                    (* impossible cases *)
     | pt1::pt2::nil ⇒                            (* diameter case *)
       if clean s then (1,m_clean s) else (2,m_dirty s)
     | pt1::pt2::pt3::nil ⇒                       (* triangle case *)
       if clean s then (3,m_clean s) else (4,m_dirty s)
     | _ ⇒                                         (* general case *)
       if clean s then (5,m_clean s) else (6,m_dirty s)
 end
end.

Definition lt_config x y :=
      Lexprod.lexprod lt lt (measure (!! x)) (measure (!! y)).
```

Conclusion About Pactole

- ⊕ Designed for mobile robot swarms
- ⊕ Ease of use for specification
- ⊕ Broadly applicable

Conclusion About Pactole

- ⊕ Designed for mobile robot swarms
  -  cycle built-in
  - Other features are possible memory, battery, ...
- ⊕ Ease of use for specification
- ⊕ Broadly applicable

Conclusion About Pactole

- $\oplus$ Designed for mobile robot swarms
  -  cycle built-in
  - Other features are possible memory, battery, . . .
- $\oplus$ Ease of use for specification
  - Expressive logic (Coq)
  - Maths can be directly expressed
  - Principled bottom-up approach
- $\oplus$ Broadly applicable

Conclusion About Pactole

$\oplus$ Designed for mobile robot swarms

-  cycle built-in
- Other features are possible — memory, battery, ...

$\oplus$ Ease of use for specification

- Expressive logic (Coq)
- Maths can be directly expressed
- Principled bottom-up approach

$\oplus$ Broadly applicable

- Highly parametric — space, sensors, execution model, ...
- Very expressive
- Common base of definition (no more mismatches)

$\Rightarrow$ A junction point for several formal results?

Conclusion About Pactole

- ⊕ Designed for mobile robot swarms
  -  cycle built-in
  - Other features are possible — memory, battery, . . .
- ⊕ Ease of use for specification
  - Expressive logic (Coq)
  - Maths can be directly expressed
  - Principled bottom-up approach
- ⊕ Broadly applicable
  - Highly parametric — space, sensors, execution model, . . .
  - Very expressive
  - Common base of definition (no more mismatches)

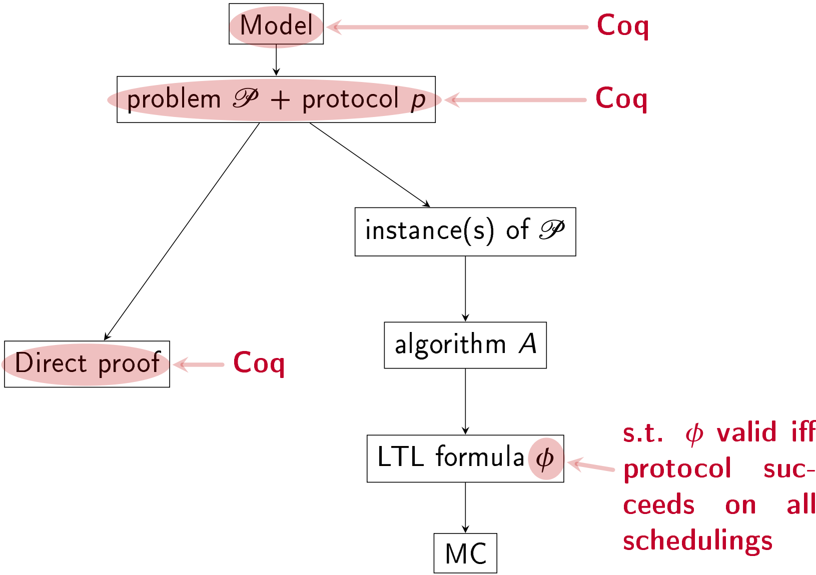
⇒ A junction point for several formal results?

- ⊖ Caveat
  - No fully automated procedure (yet)
  - Building proofs is a lot of work

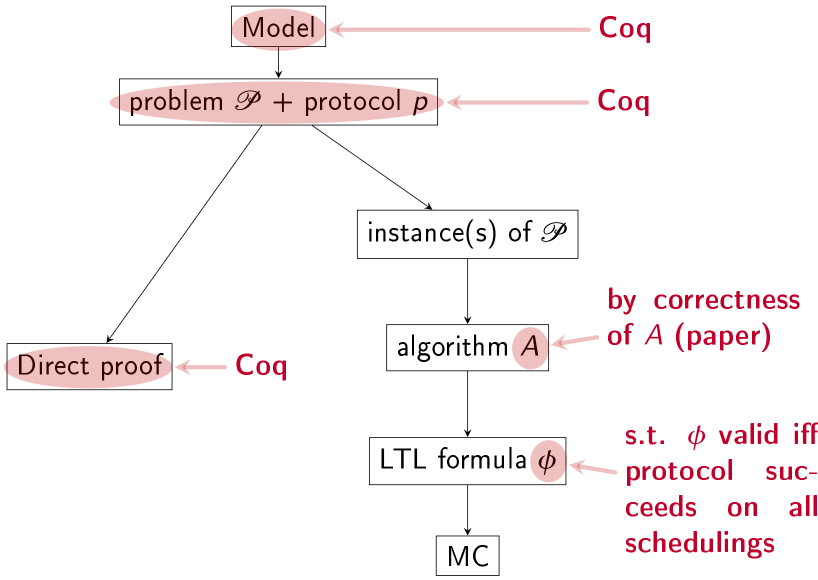
Future

- ANR SAPPORO (2019-2023):
  - Randomized protocols
  - Automated proofs (graph rewriting)
  - Non euclidean geometry (non-perfect sensors)
  - Luminous robots
- Link with model checkers

Appendix: Link with model checking



Appendix: Link with model checking



Appendix: Link with model checking