Proving in Coq (Pactole): A user perspective

David Ilcinkas
Joint work with Sébastien Bouchard

CNRS, Bordeaux, France

GT Algorithmique Distribuée
January 17, 24 & 31, 2022

Distributed computing

Usually

- Rather small algorithms
- Importance of details
- Rather long proofs

Specificities in distributed computing

- Local algorithm vs. global behavior
- Non-determinism
  - Asynchrony
  - Dynamicity
  - Crashes
  - Byzantines
- Computing entities only have partial knowledge

Distributed computing

Usually

- Rather small algorithms
- Importance of details
- Rather long proofs

Specificities in distributed computing

- Local algorithm vs. global behavior
- Non-determinism
  - Asynchrony
  - Dynamicity
  - Crashes
  - Byzantines
- Computing entities only have partial knowledge

Confidence in distributed computing

Proofs: artifact to convince oneself of the validity of a statement

From [Lamport, How to Write a 21st Century Proof, 2012]

Proofs are still written in prose pretty much the way they were in the 17th century. [...]

Proofs are unnecessarily hard to understand, and they encourage sloppiness that leads to errors.

[Concurrent (multiprocess)] algorithms can be quite subtle and hard to get right; their correctness proofs require a degree of precision and rigor unknown to most mathematicians (and many computer scientists).

Confidence in distributed computing

Proofs: artifact to convince oneself of the validity of a statement

From [Lamport, How to Write a 21st Century Proof, 2012]

> *Proofs are still written in prose pretty much the way they were in the 17th century. [...]*
> *Proofs are unnecessarily hard to understand, and they encourage sloppiness that leads to errors.*

[Concurrent (multiprocess)] algorithms can be quite subtle and hard to get right; their correctness proofs require a degree of precision and rigor unknown to most mathematicians (and many computer scientists).

Confidence in distributed computing

Proofs: artifact to convince oneself of the validity of a statement

From [Lamport, How to Write a 21st Century Proof, 2012]

> *Proofs are still written in prose pretty much the way they were in the 17th century. [...]*
> *Proofs are unnecessarily hard to understand, and they encourage sloppiness that leads to errors.*
>
> *[Concurrent (multiprocess)] algorithms can be quite subtle and hard to get right; their correctness proofs require a degree of precision and rigor unknown to most mathematicians (and many computer scientists).*

Detecting and preventing issues

Possible research directions to improve confidence

- Distributed decision / Distributed verification

  cf. Laurent Feuilloley, self-stabilization
- Experiments

  Difficult to tackle non-determinism
- Model-checking
- Certification via a proof assistant

Formal verification / model-checking

Model-checking

Automated verification based on a formalization of

- the system model
- the algorithm
- the properties to be satisfied

Pros:

- Automatic verification
- If verification fails, generally gives a counter-example
- May sometimes be able to synthesize algorithms

Cons:

- Subject to decidability and tractability issues

Formal verification / model-checking

Model-checking

Automated verification based on a formalization of

- the system model
- the algorithm
- the properties to be satisfied

Pros:

- Automatic verification
- If verification fails, generally gives a counter-example
- May sometimes be able to synthesize algorithms

Cons:

- Subject to decidability and tractability issues

Formal verification / model-checking

Model-checking

Automated verification based on a formalization of

- the system model
- the algorithm
- the properties to be satisfied

Pros:

- Automatic verification
- If verification fails, generally gives a counter-example
- May sometimes be able to synthesize algorithms

Cons:

- Subject to decidability and tractability issues

Certification

Proof assistant

- Also called Interactive theorem prover
- Checks whether a given proof of a given statement is correct
- Provides an interactive proof editor

Pros:

- More or less as powerful as paper proofs
- Provides some automation

Cons:

- Mainly manual
- Sometimes tedious even for simple proofs

Certification

Proof assistant

- Also called Interactive theorem prover
- Checks whether a given proof of a given statement is correct
- Provides an interactive proof editor

Pros:

- More or less as powerful as paper proofs
- Provides some automation

Cons:

- Mainly manual
- Sometimes tedious even for simple proofs

Certification

Proof assistant

- Also called Interactive theorem prover
- Checks whether a given proof of a given statement is correct
- Provides an interactive proof editor

Pros:

- More or less as powerful as paper proofs
- Provides some automation

Cons:

- Mainly manual
- Sometimes tedious even for simple proofs

Big (french) projects about DC in Coq

PADEC

- Dedicated to self-stabilizing algorithms
- Complex algorithms
- Precise analyses, including time complexities
- K. Altisen, P. Corbineau, S. Devismes

PACTOLE

- Dedicated to distributed computing by mobile robots
- Large variety of models and parameters
- **Positive** and **negative** results
- T. Balabonski, P. Courtieu, L. Rieg, S. Tixeuil, X. Urbain, ...

Problem: Ring exploration with stop

Model/context

- Team of robots
  - sensing the environment by taking a snapshot of it
  - that do not communicate
  - that are anonymous and oblivious
- Anonymous unoriented rings.

Goal: exploration with stop

- Each node must be visited by at least one robot.
- All robots must stop after finite time.

The Look-Compute-Move cycle

Look

The robot takes an instantaneous egocentric snapshot of the network and its robots, with strong multiplicity detection (exact number of robots at each node).

Compute

Based on this observation, it decides to stay idle or to move to some neighboring node.

Move

In the latter case it instantaneously moves towards its destination.

The Look-Compute-Move cycle

Look

The robot takes an instantaneous egocentric snapshot of the network and its robots, with strong multiplicity detection (exact number of robots at each node).

Compute

Based on this observation, it decides to stay idle or to move to some neighboring node.

Move

In the latter case it instantaneously moves towards its destination.

The Look-Compute-Move cycle

Look

The robot takes an instantaneous egocentric snapshot of the network and its robots, with strong multiplicity detection (exact number of robots at each node).

Compute

Based on this observation, it decides to stay idle or to move to some neighboring node.

Move

In the latter case it instantaneously moves towards its destination.

Identical oblivious semi-synchronous robots

Identical

Robots have no IDs. They execute the same program.

Oblivious

The robots have no memory of observations, computations and moves made in previous cycles.

Semi-Synchronous

Look/Compute/Move cycles are synchronized, but robots may sleep during some cycles.

Identical oblivious semi-synchronous robots

Identical

Robots have no IDs. They execute the same program.

Oblivious

The robots have no memory of observations, computations and moves made in previous cycles.

Semi-Synchronous

Look/Compute/Move cycles are synchronized, but robots may sleep during some cycles.

Identical oblivious semi-synchronous robots

Identical

Robots have no IDs. They execute the same program.

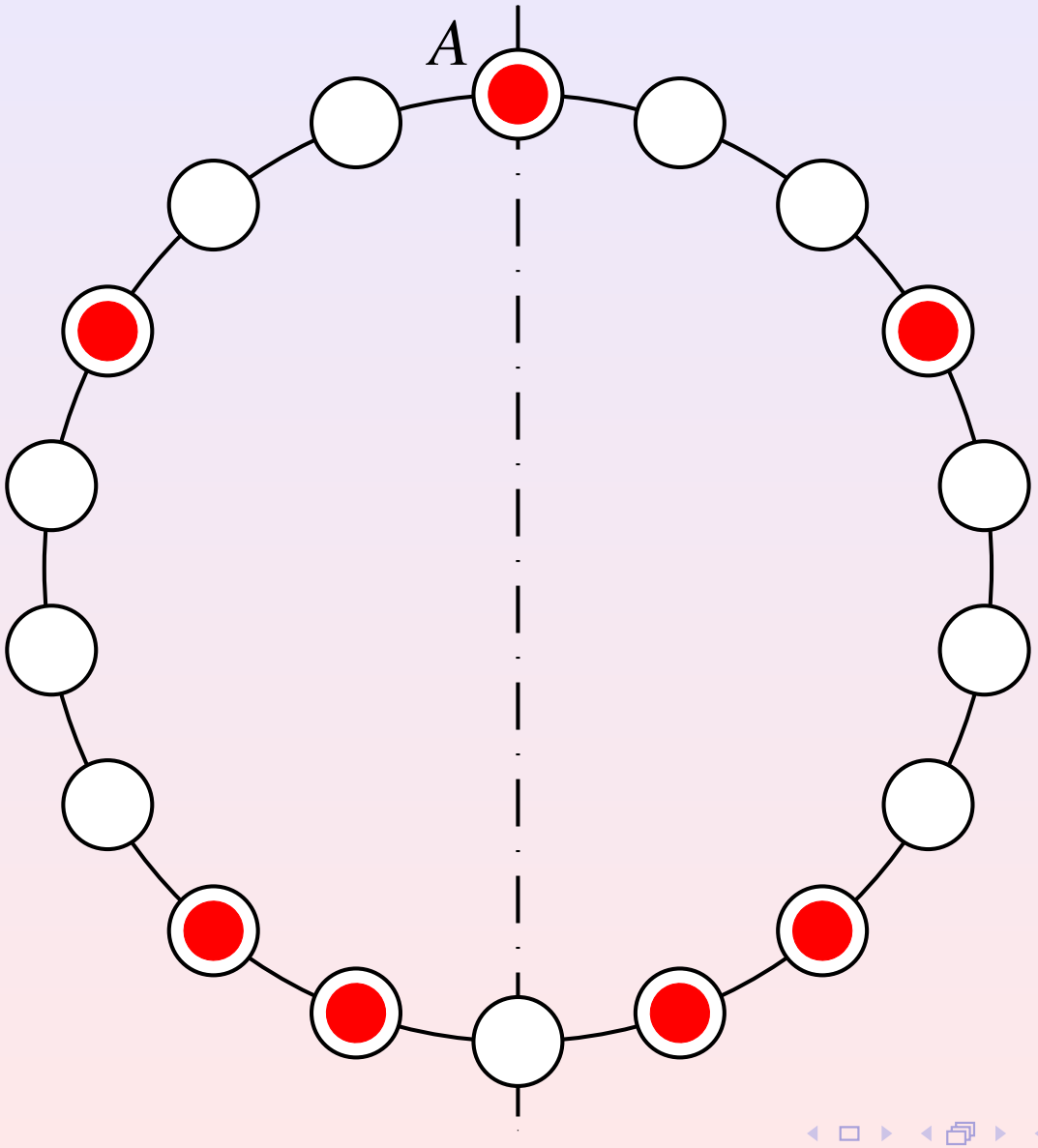
Oblivious

The robots have no memory of observations, computations and moves made in previous cycles.

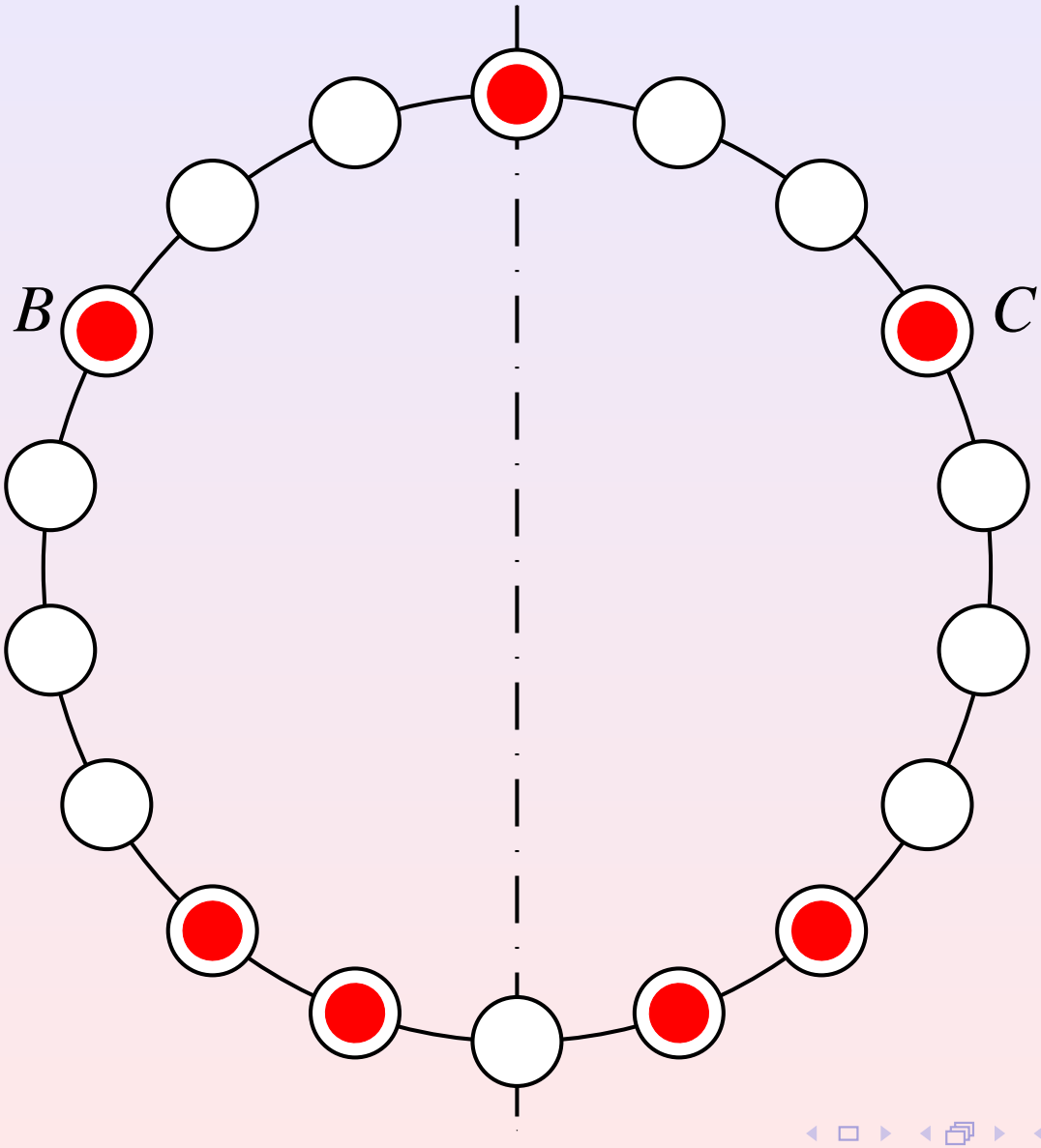
Semi-Synchronous

Look/Compute/Move cycles are synchronized, but robots may sleep during some cycles.

Some precisions



Some precisions



Problem definitions

Goal: exploration with stop

- Each node must be visited by at least one robot.
- All robots must stop after finite time.

Definition: Explo(k, n)

We say that exploration of a n-node ring is possible with $k < n$ robots if there exists an algorithm enabling the robots to perform exploration with stop starting from any initial configuration of the k robots without multiplicity (at most one robot per node).

More formal definition: Explo(k, n), with $k < n$

exists Algo, forall Adv, Config,
if is_cycle(n,Config) and has_robots(k,Config) and is_flat(Config)
then exploStop(Algo, Adv, Config)

Problem definitions

Goal: exploration with stop

- Each node must be visited by at least one robot.
- All robots must stop after finite time.

Definition: Explo(k, n)

We say that exploration of a n-node ring is possible with $k < n$ robots if there exists an algorithm enabling the robots to perform exploration with stop starting from any initial configuration of the k robots without multiplicity (at most one robot per node).

More formal definition: Explo(k, n), with $k < n$

exists Algo, forall Adv, Config,
if is_cycle(n,Config) and has_robots(k,Config) and is_flat(Config)
then exploStop(Algo, Adv, Config)

Problem definitions

Goal: exploration with stop

- Each node must be visited by at least one robot.
- All robots must stop after finite time.

Definition: Explo(k, n)

We say that exploration of a n-node ring is possible with $k < n$ robots if there exists an algorithm enabling the robots to perform exploration with stop starting from any initial configuration of the k robots without multiplicity (at most one robot per node).

More formal definition: Explo(k, n), with $k < n$

exists Algo, forall Adv, Config,
if is_cycle(n,Config) and has_robots(k,Config) and is_flat(Config)
then exploStop(Algo, Adv, Config)

Results in PACTOLE

Already in PACTOLE

- not Explo($\mathbf{1}, n$)
- not Explo(k, n) **if** k **divides** n

Our "new" result

for every positive integer m,

not Explo(k, n) $\Longrightarrow$ **not Explo**($k * m, n * m$)

Results in PACTOLE

Already in PACTOLE

- not Explo($\mathbf{1}, n$)
- not Explo(k, n) **if** k **divides** n

Our "new" result

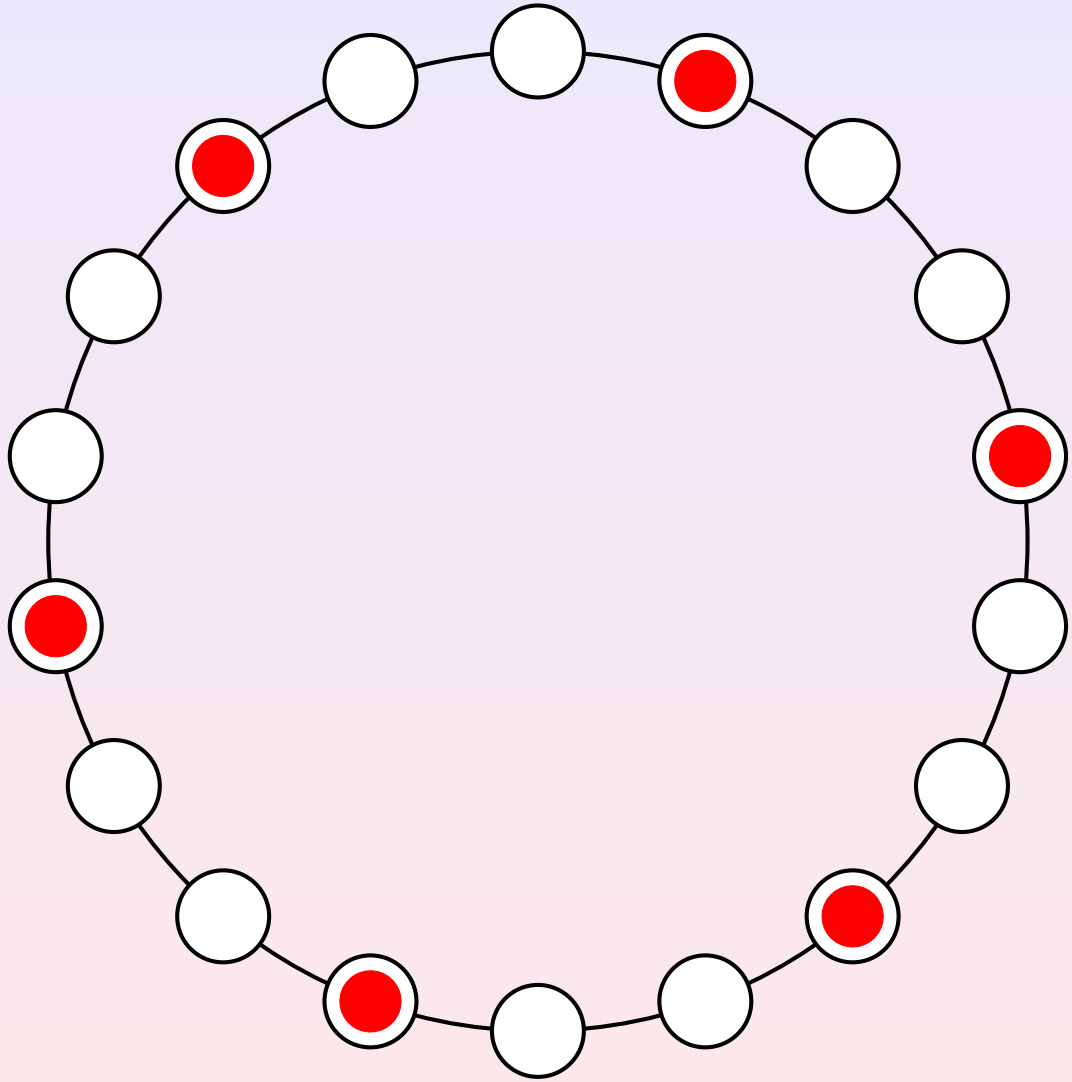
for every positive integer m,

not Explo$(k, n) \Longrightarrow$ **not Explo**$(k * m, n * m)$

When k divides n

Lemma

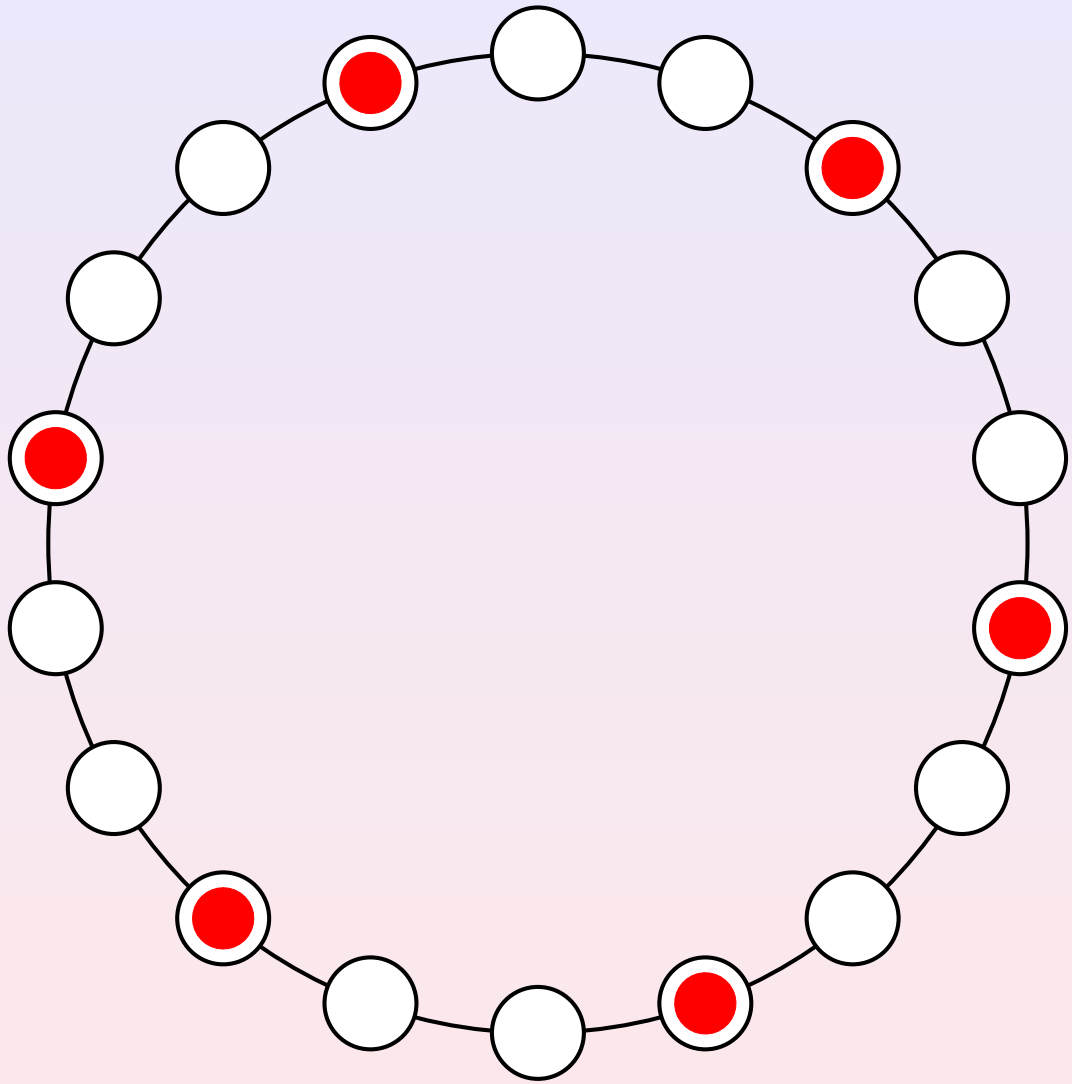
Impossible to stop (and sometimes to explore) when $k|n$.



When k divides n

Lemma

Impossible to stop (and sometimes to explore) when $k|n$.

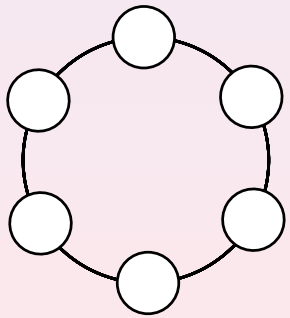


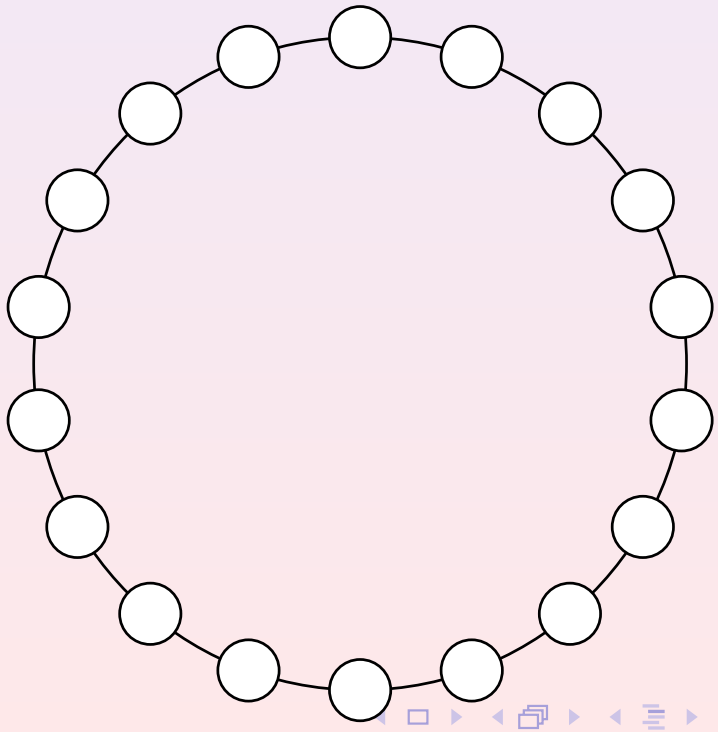
Key ingredient: expansion

Proved via: $\text{Explo}(k * m, n * m) \Longrightarrow \text{Explo}(k, n)$

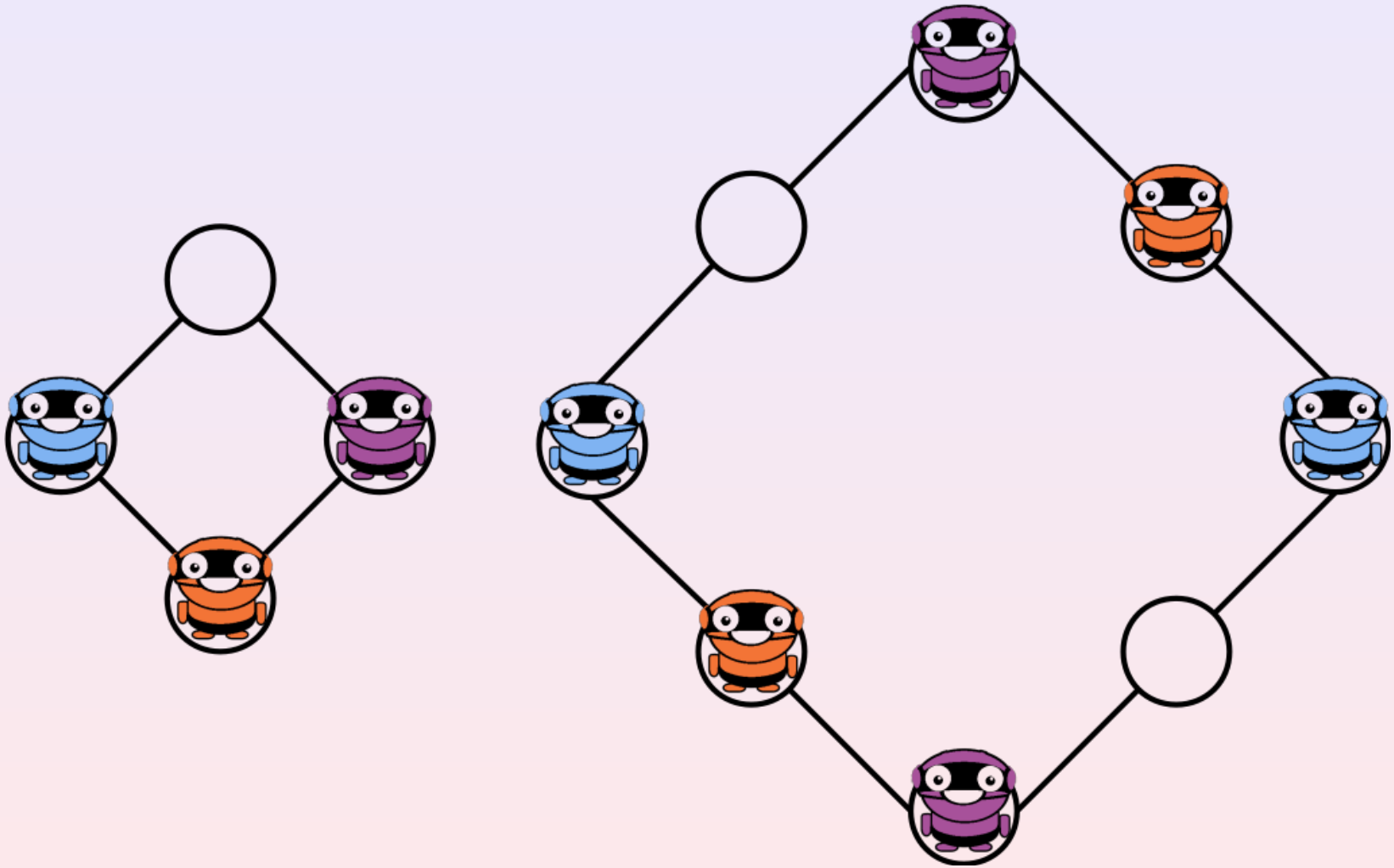
Transformations between small _1 and big _m

- From _1 to _m: $i \rightarrow \{n \text{ or } k\} * j + i$, for $0 \leq j < m$
- From _m to _1: $i \rightarrow i$ modulo $\{n \text{ or } k\}$

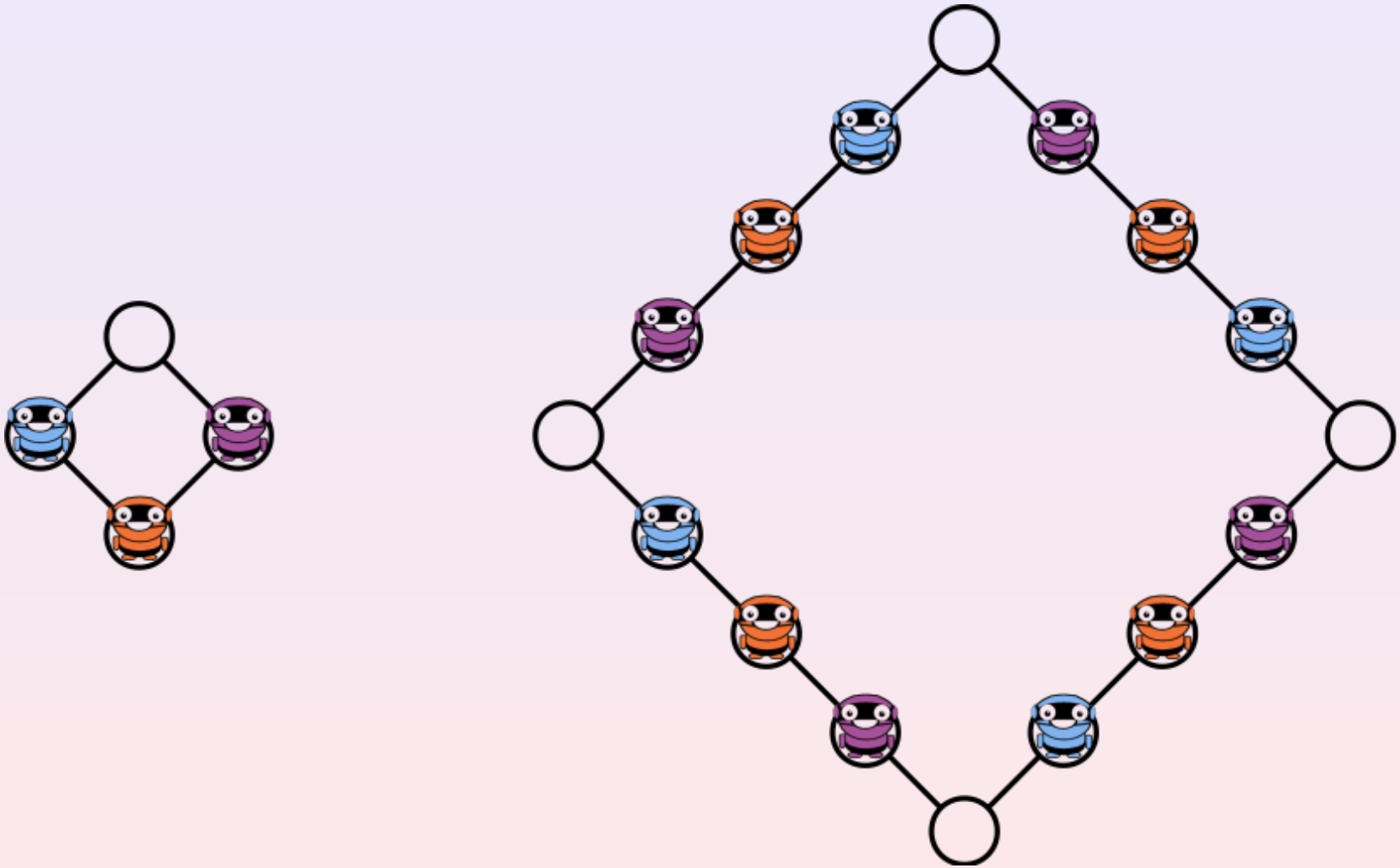




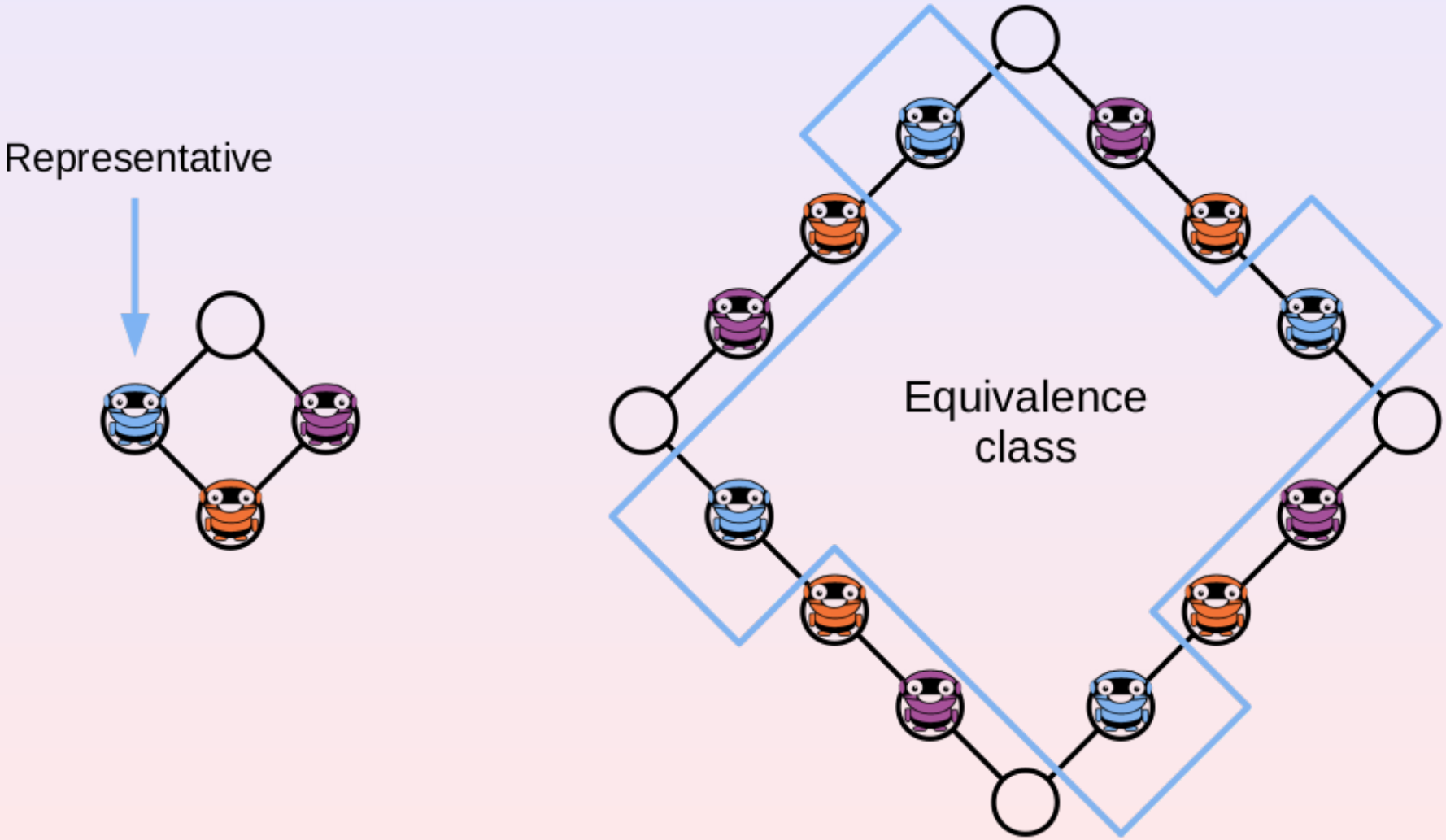
“Expanding” a “small” configuration



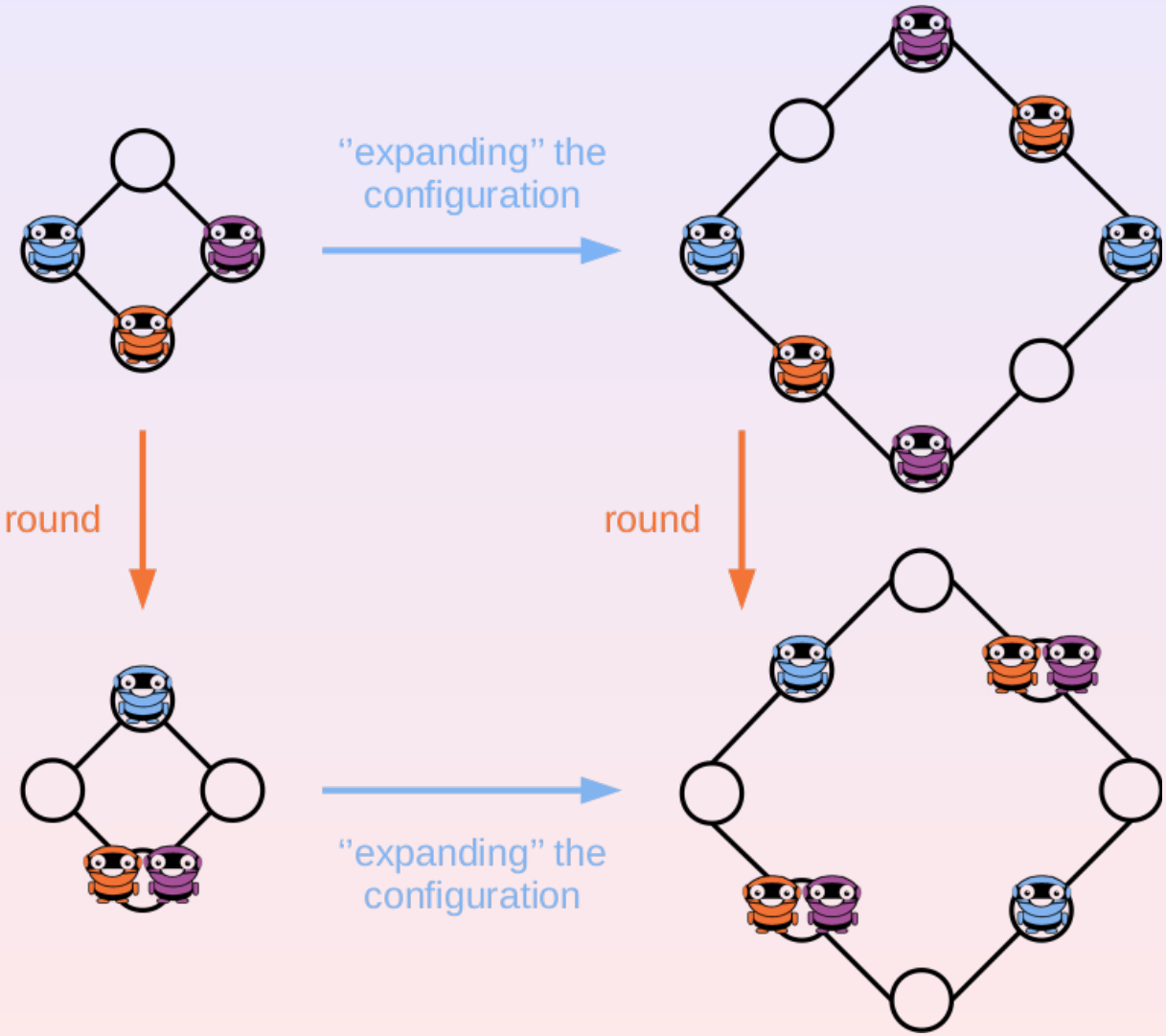
“Expanding” a “small” configuration



“Expanding” a “small” configuration



“Similarity” of the two executions



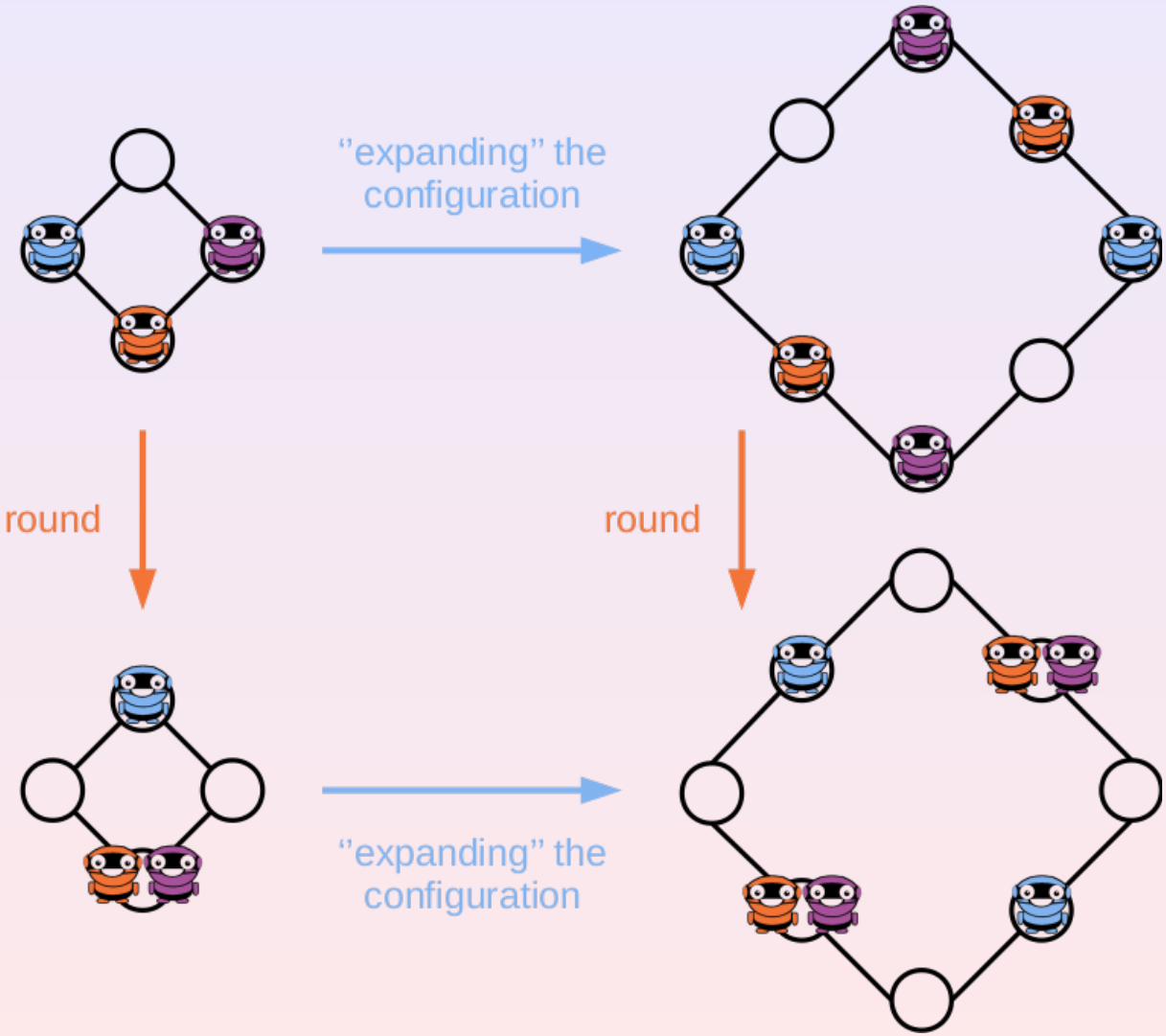
Sketch of proof (1/2)

1. Let $\mathcal{I}_m$ and $\mathcal{I}_1$ be any "corresponding big and small instances".
2. We assume that there exists an algorithm $\mathcal{A}_m$ which achieves exploration with stop in $\mathcal{I}_m$ regardless of the demon and of the initial configuration.
3. Let d_1 be any demon that $\mathcal{A}_1$ could face when trying to explore $\mathcal{I}_1$ and let c_1 be any initial configuration for the execution of $\mathcal{A}_1$.
4. Let c_m be the "expansion" of c_1.
5. Let d_m be the demon which at any round, for any class c, activates all the robots of c iff d_1 activates the representative of c
6. Algorithm $\mathcal{A}_m$ achieves exploration with stop in particular when facing d_m from the initial configuration c_m.

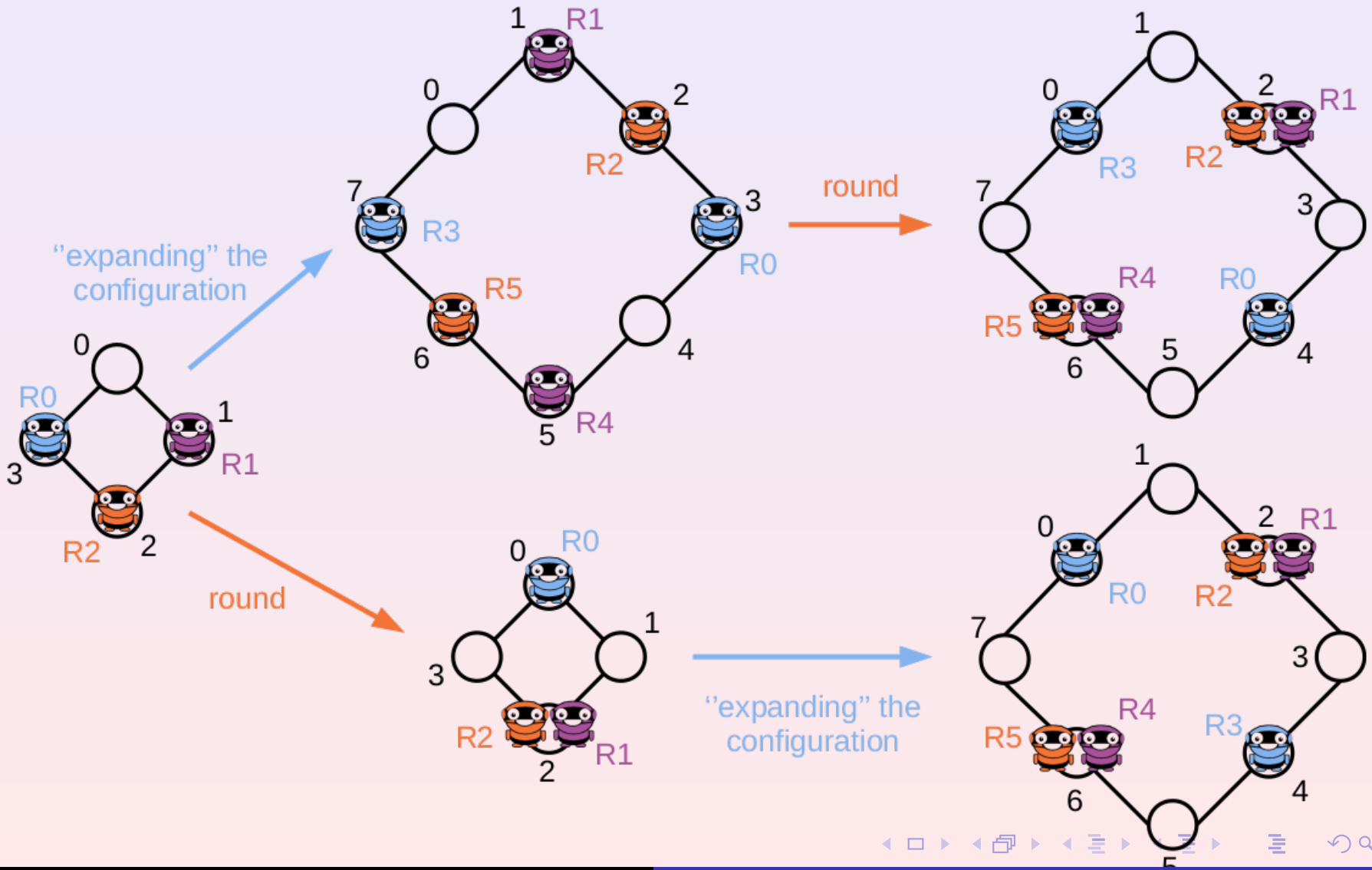
Sketch of proof (2/2)

7. We build the algorithm $\mathcal{A}_1$ which gives the same instructions as $\mathcal{A}_m$ facing d_m from c_m but only to the robots which appear in $\mathcal{I}_1$.
8. Let $\mathcal{E}_m$ be the execution of $\mathcal{A}_m$ facing d_m from c_m.
9. Let $\mathcal{E}_1$ be the execution of $\mathcal{A}_1$ facing d_1 from c_1.
10. Let $\mathcal{E}'_m$ be the execution composed of the "expansions" of the configurations of $\mathcal{E}_1$.
11. We prove that $\mathcal{E}_m$ and $\mathcal{E}'_m$ are "the same".
12. We deduce that $\mathcal{A}_1$ achieves exploration with stop in $\mathcal{I}_1$ regardless of the demon and of the initial configuration.

“Similarity” of the two executions



"Similarity" of the two executions



Challenges

1. Coq basics: definitions (Gallina) and proofs (tactics)
2. Constructive logics
3. Setoids (sets equipped with an equivalence relation)
4. Type classes (allows to make parameters implicit)
5. Induction and coinduction
6. Trying to prove statements that are indeed true!
7. Prove all the
   - Clearly [...]
   - It is easy to see that [...]
   - Similarly, [...]

Challenges

1. Coq basics: definitions (Gallina) and proofs (tactics)
2. Constructive logics
3. Setoids (sets equipped with an equivalence relation)
4. Type classes (allows to make parameters implicit)
5. Induction and coinduction
6. Trying to prove statements that are indeed true!
7. Prove all the
   - Clearly [...]
   - It is easy to see that [...]
   - Similarly, [...]

Challenges

1. Coq basics: definitions (Gallina) and proofs (tactics)
2. Constructive logics
3. Setoids (sets equipped with an equivalence relation)
4. Type classes (allows to make parameters implicit)
5. Induction and coinduction
6. Trying to prove statements that are indeed true!
7. Prove all the
   - Clearly [...]
   - It is easy to see that [...]
   - Similarly, [...]

Challenges

1. Coq basics: definitions (Gallina) and proofs (tactics)
2. Constructive logics
3. Setoids (sets equipped with an equivalence relation)
4. Type classes (allows to make parameters implicit)
5. Induction and coinduction
6. Trying to prove statements that are indeed true!
7. Prove all the
   - Clearly [...]
   - It is easy to see that [...]
   - Similarly, [...]

Challenges

1. Coq basics: definitions (Gallina) and proofs (tactics)
2. Constructive logics
3. Setoids (sets equipped with an equivalence relation)
4. Type classes (allows to make parameters implicit)
5. Induction and coinduction
6. Trying to prove statements that are indeed true!
7. Prove all the
   - Clearly [...]
   - It is easy to see that [...]
   - Similarly, [...]

Challenges

1. Coq basics: definitions (Gallina) and proofs (tactics)
2. Constructive logics
3. Setoids (sets equipped with an equivalence relation)
4. Type classes (allows to make parameters implicit)
5. Induction and coinduction
6. Trying to prove statements that are indeed true!
7. Prove all the
   - Clearly [...]
   - It is easy to see that [...]
   - Similarly, [...]

Challenges

1. Coq basics: definitions (Gallina) and proofs (tactics)
2. Constructive logics
3. Setoids (sets equipped with an equivalence relation)
4. Type classes (allows to make parameters implicit)
5. Induction and coinduction
6. Trying to prove statements that are indeed true!
7. Prove all the
   - Clearly [...]
   - It is easy to see that [...]
   - Similarly, [...]

Some facts/numbers

Preliminary expertise

- David: The Coq working group (since 2018) and PADEC
- Sébastien: No experience with Coq

Total of 1800 lines of code

- 200 lines of definitions
- 400 lines related to Setoids (compatibility)
- 200 lines for Exploration and Stop properties
- 1000 lines for the "easy" commutativity lemma and the other "obvious" things

Duration: 3 months, a few hours a week

Some facts/numbers

Preliminary expertise

- David: The Coq working group (since 2018) and PADEC
- Sébastien: No experience with Coq

Total of 1800 lines of code

- 200 lines of definitions
- 400 lines related to Setoids (compatibility)
- 200 lines for Exploration and Stop properties
- 1000 lines for the "easy" commutativity lemma and the other "obvious" things

Duration: 3 months, a few hours a week

Some facts/numbers

Preliminary expertise

- David: The Coq working group (since 2018) and PADEC
- Sébastien: No experience with Coq

Total of 1800 lines of code

- 200 lines of definitions
- 400 lines related to Setoids (compatibility)
- 200 lines for Exploration and Stop properties
- 1000 lines for the "easy" commutativity lemma and the other "obvious" things

Duration: 3 months, a few hours a week

A final note from L. Lamport

Quote about the Temporal Logic of Actions

TLA does have the following disadvantages:

- It can describe only a real algorithm, not a vague, incomplete sketch of an algorithm.
- You can specify an algorithm's correctness condition in TLA only if you understand what the algorithm is supposed to do.
- TLA makes it harder to cover gaps in a proof with handwaving.

Some researchers may find these drawbacks insurmountable.

By the way, Amazon started to use formal specification and model checking in 2011, year of a major disruption in their systems...