Synchronous Gathering without Multiplicity Detection: a Certified Algorithm

Thibaut Balabonski, Amélie Delga, Lionel Rieg, Sébastien Tixeuil, Xavier Urbain

Noname manuscript No.
(will be inserted by the editor)

Synchronous Gathering without Multiplicity Detection: a Certified Algorithm

Thibaut Balabonski · Amélie Delga · Lionel Rieg · Sébastien Tixeuil · Xavier Urbain

the date of receipt and acceptance should be inserted later

Abstract In mobile robotic swarms, the gathering problem consists in coordinating all the robots so that in finite time they occupy the same location, not known beforehand. Multiplicity detection refers to the ability to detect that more than one robot can occupy a given position. When the robotic swarm operates synchronously, a well-known result by Cohen and Peleg permits to achieve gathering, provided robots are capable of multiplicity detection.

We present a new algorithm for synchronous gathering, that does *not* assume that robots are capable of multiplicity detection, nor make any other extra assumption. Unlike previous approaches, the correctness of our proof is certified in the model where the protocol is defined, using the Coq proof assistant.

1 Introduction

Networks of mobile robots have captured the attention of the distributed computing community, as they promise new applications (rescue, exploration, surveillance) in

Thibaut Balabonski
LRI, Univ. Paris-Sud, CNRS, Université Paris-Saclay, France

Amélie Delga
École Nat. Sup. d'Informatique pour l'Industrie et l'Entreprise (ENSIIE), Évry, France

Amélie Delga · Sébastien Tixeuil
UPMC Sorbonne Universités, LIP6-CNRS 7606, France

Lionel Rieg
Collège de France, Paris, France

Lionel Rieg
Yale University, New Haven, CT, USA

Sébastien Tixeuil
Institut Universitaire de France, Paris, France

Xavier Urbain
Université Claude Bernard Lyon-1, LIRIS CNRS UMR 5205, Université de Lyon, France

potentially dangerous (and harmful) environments. Since its initial presentation [23], this computing model has grown in popularity[1] and many refinements have been proposed (see [17] for a recent state of the art). From a theoretical point of view, the interest lies in characterising the exact conditions for solving a particular task.

A computing model for mobile robots. In the model we consider, robots operate in Look-Compute-Move cycles. In each cycle a robot "Looks" at its surroundings and obtains (in its own coordinate system) a snapshot containing some information about the locations of all robots. Based on this visual information, the robot "Computes" a destination location (still in its own coordinate system) and then "Moves" towards the computed location. When the robots are oblivious, the computed destination in each cycle depends only on the snapshot obtained in the current cycle (and not on the past history of execution). The snapshots obtained by the robots are not necessarily consistently oriented in any manner.

The model of execution impacts significantly the solvability of collaborative tasks. Three different levels of synchronisation have been considered. Among these, the strongest model [23] is the fully synchronised (FSYNC) model where each stage of each cycle is performed simultaneously by all robots. At the other end of the spectrum, the asynchronous model [17] (ASYNC) allows for arbitrary delays between the Look, Compute and Move stages and the movement itself may take an arbitrary amount of time, possibly a different amount for each robot. In the semi-synchronous (SSYNC) model [23], which lies somewhere between the two extreme models, time is discretised into rounds and in each round an arbitrary subset of the robots are active. The active robots in a round perform exactly one atomic Look-Compute-Move cycle in that round. It is assumed that the scheduler (seen as an adversary) is fair in the sense that it guarantees that in any configuration, any robot is activated within a finite number of rounds.

Furthermore, the scheduler has the ability to stop a robot before it has completed its move, provided the robot has already travelled some positive distance δ. If a robot wants to move by some distance $d < \delta$, the scheduler cannot stop its movement. The value of δ is unknown to the robots, and is just meant to prevent the scheduler to make them move by infinitely small distances. These stoppable moves are referred to as *flexible* moves in the remainder of the paper, as opposed to *rigid* moves where the computed destination is always reached.

The gathering problem. The gathering problem is one of the benchmarking tasks in mobile robot networks, and has received a considerable amount of attention (see [17, 3] and references herein). The gathering task consists in making all robots (considered as dimensionless points in a two dimensional Euclidean space) reach a single point, not known beforehand, in finite time. A foundational result [23] shows that in the SSYNC model, no oblivious deterministic algorithm can solve gathering for two robots. This result can be extended [13] to the bivalent case, that is, when an even number of robots is initially split evenly in exactly two locations. In general,

[1] The 2016 SIROCCO Prize for Innovation in Distributed Computing was awarded to Masafumi Yamashita for this line of work.

without extra assumptions in the execution model (*e.g.* a common coordinate system, persistent memory, the ability to detect multiple robots at a given location, use of probabilistic variables, etc.), it is impossible to solve gathering [19] for any set of at least two robots in the SSYNC model. As all possible executions in SSYNC are also possible in ASYNC, those impossibilities also hold in ASYNC. Hence, the only possibility to solve gathering without extra assumptions is to consider the FSYNC model.

Cohen and Peleg [11] proposed the *centre of gravity* (*a.k.a.* CoG) algorithm (the robots aim for the location that is the barycentre of all observed robot locations) for the purpose of convergence (a weaker requirement than gathering, which mandates robots to reach locations that are arbitrarily close to one another) in the SSYNC model. They demonstrate that for the FSYNC model, their algorithm actually solves gathering since all robots eventually become closer than δ from the barycentre, and hence all reach it in the next round.

However, the CoG algorithm does not prevent more than one robot to occupy the exact same location before gathering, even if they start from distinct locations. For example, consider two robots r_1 and r_2 aligned toward the barycentre at some round, at respective distances d_1 and d_2 ($d_1 < d_2$) that are both greater than δ. Then, the scheduler stops r_1 after δ and r_2 at the same location. Robots r_1 and r_2 now occupy the same location. One immediate consequence of this observation is that in the next round, to compute the barycentre, observing robots must take into account both r_1 and r_2. That is, using the CoG algorithm, robots must make use of *multiplicity detection*, *i.e.* be able to detect how many robots occupy simultaneously a given location.

Overall, the question of gathering feasibility in FSYNC without multiplicity detection (nor any other additional assumption) remained open.

Formal methods for mobile robots. Designing and proving mobile robot protocols is notoriously difficult. Formal methods encompass a long-lasting path of research that is meant to overcome errors of human origin. Unsurprisingly, this mechanised approach to protocol correctness was successively used in the context of mobile robots [7,15,5,2,18,13,8,20].

Model-checking proved useful to find bugs in existing literature [5,16] and to assess formally published algorithms [15,5,20], in a simpler setting where robots evolve in a *discrete space* where the number of possible locations is finite. Automatic program synthesis (for the problem of perpetual exclusive exploration in a ring-shaped discrete space) is due to Bonnet *et al.* [7], and can be used to obtain automatically algorithms that are "correct-by-design". The approach was refined by Millet *et al.* [18] for the problem of gathering in a discrete ring network. As all aforementioned approaches are designed for a discrete setting where both the number of locations and the number of robots are known, they cannot be used in the continuous space where the robots locations take values in a set that is not enumerable, and they cannot permit to establish results that are valid for any number of robots. The recent attempt to parameterised model checking by Sangnier *et al.* [22] yielded mostly negative results: reachability property are in general undecidable even when the number of robots is fixed and only the network size is parameterised (a ring in

their case). Overall, model-checking approaches do not yet permit to tackle problems that involve arbitrarily many robots evolving in a bidimensional Euclidean space.

The use of a mechanical proof assistant like Coq[2] allows for more genericity as this formal proof based approach is not limited to *particular instances* of algorithms. Recent uses of Coq in Distributed Computing include that of Castéran *et al.* [9], who use Coq and their library Loco to prove positive and negative results about subclasses of LC systems, and that of Altisen *et al.* [1], who provide a Coq framework to study self-stabilising algorithms.

Developed for the Coq proof assistant, the Pactole[3] framework enabled the use of high-order logic to certify impossibility results [2] for the problem of convergence: for any positive ε, robots are required to reach locations that are at most ε apart. Another classical impossibility result that was certified using the Pactole framework is the impossibility of gathering starting from a bivalent configuration [13]. Recently, positive certified results for SSYNC gathering with multiplicity detection were provided by Courtieu *et al.* [14].

Our contribution. We propose a protocol for oblivious mobile robot gathering in FSYNC that does not require multiplicity detection (nor any other extra assumption). Our protocol, called CoGiL (for Centre of Gravity of inhabited Locations), is derived from CoG as follows: robots aim to the barycentre of observed *occupied* locations (that is, without considering how many robots occupy a given location). We also present a proof of correctness for our CoGiL protocol.

Unlike previous approaches, our proof is *certified* in the model where the protocol is defined, using the Coq proof assistant. Throughout this paper, links to the Coq development are denoted by a ⇝ symbol in the margin. The sources package is available at `http://pactole.lri.fr`, as well as its online `html` documentation.

Roadmap. Section 2 describes our Pactole formal framework for mobile robots in Coq, while our cases studies are developed in Section 3, including their formal proof of correctness. Section 4 gives some insights about the benefits of our methodology for mobile robot protocol design.

2 A Formal Model to Prove Robot Protocols

To certify results and to guarantee the soundness of theorems, we use Coq, a Curry-Howard-based interactive proof assistant enjoying a trustworthy kernel. The (functional) language of Coq is a very expressive λ-calculus: the *Calculus of Inductive Constructions* (CIC) [12]. In this context, datatypes, objects, algorithms, theorems and proofs can be expressed in a unified way, as terms.

The reader will find in [6] a very comprehensive overview and good practices with reference to Coq. Developing a proof in a proof assistant may nonetheless be tedious, or require expertise from the user. To make this task easier, we are actively developing

[2] `http://coq.inria.fr`
[3] `http://pactole.lri.fr`

(under the name Pactole) a formal model, as well as lemmas and theorems, to specify and certify results about networks of autonomous mobile robots. It is designed to be robust and flexible enough to express most of the variety of assumptions in robots network, for example with reference to the considered space: discrete or continuous, bounded or unbounded...

We do not expect the reader to be an expert in Coq but of course the specification of a model for mobile robots in Coq requires some knowledge of the proof assistant. We want to stress that the framework eases the developer's task in two main directions. Firstly, it clearly separates the specification and proof phases. This allows for non-expert users of the Coq system to write easily specifications and theorem statements in the Coq language, which is very expressive and close to usual mathematical writing, while still ensuring that every concept is precisely and formally defined, without any implicit information. The proving phases requires more knowledge of the Coq proof assistant but, being clearly separated from the specification phase, does not need to be performed by the same person. Secondly, the framework is built from modular components like the kind of space, the sensor capabilities of robots, the various hypotheses of synchronicity or fairness, or the existence of Byzantine faults. These components can be combined together to create the adequate setting for the user but also to prove generic results in each component. The notations and definitions we give hereafter should be simply read as typed functional expressions.

The Pactole model has been sketched in [2, 13]; we recall its main characteristics.

We use two important features of Coq:

1. a formalism of *higher-order* logic to quantify over programs, demons, etc., which allows us to prove generic results, and
2. the possibility to define *inductive* and *coinductive* types [21] to express inductive and coinductive datatypes and properties.

Coinductive types are in particular of invaluable help to express infinite behaviours, infinite datatypes and properties on them, as we shall see with demons.

Robots are anonymous, however proofs sometimes need to identify some of them, or to split them into separate groups. Thus, we consider given a finite set of *identifiers*, isomorphic to a segment of $\mathbb{N}$. If relevant, it is convenient to distinguish between identifiers of Byzantine robots, of the form `Byz b` for `b` a name, and identifiers of correct robots, of the form `Good g` for `g` a name. In this work, we do not consider Byzantine robots (although the Pactole framework allows for them) and we hereafter omit the set `G` of names of correct robots unless it is necessary to characterise the number of robots. Robots are distributed in space, at places called *locations*. We call a *configuration* a *function* from the set of identifiers to the space of locations. For instance, the location of a robot with identifier `id` in a configuration `conf` is simply obtained by the application `(conf id)`.

In this definition of configurations, all robots are still identified as we can get the location of a robot from its identifier. In particular, equality between configurations does *not* boil down to the equality of the multisets of inhabited locations. In order to ensure that robots are anonymous and indistinguishable, we have to make sure that the embedded algorithm cannot make use of those identifiers.

Spectrum. The computation of any robot's target location is based on the perception it gets from its environment, that is, in an FSYNC execution scheme, from a configuration. The result of this observation may be more or less accurate, depending on the capabilities of sensors. A robot's perception of a configuration is called a *spectrum*. To allow for different assumptions to be studied, we leave abstract the type *spectrum* (`Spect.t`) and the notion of spectrum of a configuration. In addition to a datatype, a spectrum definition must contain the definition of (a decidable) equality on spectra, a conversion function (`Spect.from_config`) turning a configuration into a spectrum, a formula (`Spect.is_ok`) expressing the relation between a configuration and its spectrum, and the fact that the conversion function satisfies it:

```
∀ config, Spect.is_ok (Spect.from_config config) config
```

More precisely, the formula `Spect.is_ok` characterises the information from the configuration that is still present in the spectrum, for example it may ensure that the locations in a spectrum correspond to actual locations of robots in the relevant configuration.

```
Module Type Spect(Location : DecidableType)(N : Size).
  (* Spectra are abstract decidable types. *)
  Parameter t : Type.
  (* They are equipped with an equality relation *)
  Parameter eq : t → t → Prop.
  (* which is an equivalence relation *)
  Parameter eq_equiv : Equivalence eq.
  (* and which is decidable. *)
  Parameter eq_dec : ∀ x y : t, {eq x y} + {¬ eq x y}.

  (* Turning a configuration into a spectrum (erasing information). *)
  Parameter from_config : Config.t → t.
  (* Equal configurations give equal spectra. *)
  Declare Instance from_config_compat :
    Proper (Config.eq ⇉ eq) from_config.

  (* An abstract predicate validating spectra for a configuration.  *)
  Parameter is_ok : t → Config.t → Prop.
  (* from_config gives a correct spectrum. *)
  Parameter from_config_spec :
    ∀ config, is_ok (from_config config) config.
End Spectrum.
```

When needed, those abstract properties may be instantiated in accordance to the requirements and assumptions.

Robograms, representing protocols, will then output a location when given a spectrum (instead of a configuration), thus guaranteeing that assumptions over sensors are fulfilled. For instance, the spectrum for anonymous robots with *strong* global multiplicity detection (this capacity refers to the ability to count exactly how many robots occupy any observed location) could be the multiset of inhabited locations. In a setting where robots do not enjoy the detection of multiplicity and just know if a location is inhabited or not, the *set* of inhabited locations is a suitable spectrum.

In the following we will distinguish a *demon* configuration (resp. spectrum), which is expressed in the global frame of reference, from a *robot* configuration (resp. spectrum), which is expressed in the robot's own frame of reference. At each step of

the distributed protocol (see definition of `round` below) the demon configuration is transformed, that is recentred, mirrored, rotated, and scaled into the considered robot one before being transformed into a spectrum given as parameter to the robogram. Depending on assumptions, zoom and rotation factors may be constant or chosen by the demon at each step, shared by all robots or not, etc.

Demon for flexible movements. In the context of *flexible* movements, as defined by Suzuki and Yamashita [23], robots are not ensured to reach the location they computed as their goal during their compute phase. It is nevertheless assumed that if they do not reach their goal, a minimum (absolute) distance δ is travelled. Of course, the value of δ is *unknown* to the robots, as they are just aware that some δ exists. Hence, robots either reach their destination goal when it is at distance δ or less, or travel at least δ towards it.

Rounds in this FSYNC setting are thus characterised by each of the oblivious robots getting both its new self-centred frame of reference, and the ratio of its actual movement over (the distance to) its computed destination.

We call *demonic action* this operation together with the logical properties ensuring, for example, that new frames of reference make sense, and that the provided ratio belongs to the $[0,1]$ interval. Formally, a demonic action consists of a function `step` possibly assigning a change of referential and a ratio to each robot identifier, that satisfies properties `step_⋆`. The general definition below also foresees a relocation function for robots with Byzantine faults, which will be irrelevant here.

```
Record demonic_action := {
  relocate_byz : Names.B → Location.t;
  step : Names.ident → option (  (Location.t → Sim.t) * R  );
                            (* change of referential * travel ratio *)

  step_compat : Proper (eq ⇉ opt_eq ((Location.eq ⇉ Sim.eq) * (eq_ℝ)))
                       step;
  step_zoom   : ∀ id sim c, step id = Some sim →
                (fst sim c).(Sim.zoom) ≠ 0_ℝ;
  step_center : ∀ id sim c, step id = Some sim →
                Location.eq (fst sim c).(Sim.center) c;
  step_flexibility : ∀ id sim, step id = Some sim →
                     (0 ≤ snd sim ≤ 1)_ℝ
}.
```

Demons are streams of demonic actions. As such, they are naturally defined in Coq as a coinductive object, through the use of coinductive *streams*.

```
Definition demon := Streams.t demonic_action.
```

Synchrony constraints (e.g. fairness) may be defined as coinductive properties on demons, as detailed in [2, 13]. Although Coq can accommodate more powerful definitions when necessary, we also provide notations for the usual temporal operators for people familiar with them: for `A` a type, and `Stream.t A` the type of streams of elements of type `A`, we define

```
(* Lifting a property on the current configuration to streams *)
Definition instant {A : Type} (P : A → Prop) := fun s ⇒ P (hd s).
```

```
(* A property on execution that must hold at every step *)
CoInductive forever {A} (P : Streams.t A → Prop) (s : Streams.t A) :=
  Always : P s → forever P (tl s) → forever P s.

(* A property on execution that must hold eventually *)
Inductive eventually {A} (P : Streams.t A → Prop) (s : Streams.t A) :=
  | Now : P s → eventually P s
  | Later : eventually P (tl s) → eventually P s.
```

For instance, being FSYNC is a property of the demon, stating that the `step` function always assigns a referential and a ratio to every robot. Hence the option type returned by `step` is never the empty value `None`.

```
(* Property of being fully synchronous at the first step. *)
Definition FullySynchronousInstant : demon → Prop :=
  Streams.instant (fun da ⇒ ∀ g, step da g ≠ None).

(* A demon is fully synchronous if it is fully synchronous
   at all steps. *)
Definition FullySynchronous : demon → Prop :=
  Streams.forever FullySynchronousInstant.
```

The Pactole framework provides theorems that state the equivalence between rigid movements models and flexible models when the ratio of actual movement is always 1. Developments in a rigid context may thus be written free of cumbersome irrelevant details dealing with the movement ratio which is relevant in the flexible case but is always 1 in the rigid one.

Robogram. Robograms may be naturally defined in a *completely abstract manner*, without any concrete code, in our Coq model. They consist of an actual algorithm `pgm` that represents the considered protocol and that takes a spectrum as input and returns a location, and a compatibility property `pgm_compat` stating that target locations are the same if equivalent spectra are given (for some equivalence on spectra).

```
Record robogram := {
  pgm :> Spect.t → Location.t;
  pgm_compat : Proper (Spect.eq ⇛ Location.eq) pgm }.
```

Execution of a round. The actual location of arrival for a robot is determined by the protocol, which computes a local target from the perceived spectrum, and the demon-provided ratio which is applied to the local target to obtain a chosen target. If the distance between the robot's original location and its (demon-) chosen target is more than δ then the robot stops at the chosen target, otherwise it reaches its protocol-computed destination (local target). This concise way of proceeding ensures that either the protocol-computed destination is reached or at least δ is travelled.

```
Definition round (δ : R)
  (r : robogram) (da : demonic_action) (conf : Config.t) : Config.t :=
(* for a given robot, we compute the new configuration *)
fun id ⇒
  let loc := conf id in        (* loc is the current location of id
                                  seen by the demon                  *)
  match da.(step) id with      (* Is the robot activated?            *)
```

```
    | None ⇒ loc              (** If not activated, do nothing      *)
    | Some (sim, mv_ratio) ⇒  (** If activated
                                  with similarity [sim (conf g)]
                                  and move ratio [mv_ratio]     *)
      match id with
      | Byz b ⇒ da.(relocate_byz) b (* Byzantine robots are relocated
                                       by the demon                *)
      | Good g ⇒       (* configuration expressed in the frame of g *)
        let frame_change := sim (conf (Good g)) in
        let local_conf := Config.map frame_change conf in
                          (* apply r on spectrum                   *)
        let local_target := r (Spect.from_config local_conf) in
                          (* the demon chooses a point on the line
                             from the target by mv_ratio           *)
        let chosen_target := Location.mul mv_ratio local_target in
                          (* let's get back to demon ref           *)
        frame_change^-1
          (if   δ ≤ Location.dist (frame_change^-1 chosen_target) loc
           then chosen_target
           else local_target)
      end
end.
```

3 Centre of Gravity Algorithms

Notations. In the sequel, we denote by: C a configuration, $C(r)$ the location of Robot r in Configuration C, and S_C the global spectrum associated to C.

3.1 Centre of Gravity Algorithms Variants.

Cohen and Peleg [10,11] define the CoG algorithm as depicted in Algorithm 1. A robot simply moves toward the centre of gravity of all robots locations. Since robots may occupy the same location in space, the proper calculation of the centre of gravity implies that the robots are capable of strong global multiplicity detection: for each inhabited location, the robots can count the number of robots on that location.

Algorithm 1 Protocol CoG (for Robot r in Configuration C)

Move toward the centre of gravity of robot locations $c_{pos} = \frac{1}{|C|} \times \sum_{r \in C} C(r)$

We define the CoGiL algorithm in Algorithm 2. Here, we do not assume that robots are capable of multiplicity detection, so robots simply move toward the centre of gravity of inhabited locations. Note that the number of those inhabited locations is not necessarily monotonically decreasing: although two robots at the same location will compute the same destination (`local_target` in the definition of `round`), they might end up in different locations due to the flexibility of their respective movements.

Observe also that when robots are endowed with strong multiplicity detection, the spectrum becomes a multiset of locations. Further discussion on this issue is delegated to Section 3.3.

Algorithm 2 Protocol CoGiL (for Robot r in Configuration C)

Move toward the centre of gravity of inhabited locations $c_{pos} = \frac{1}{|S_C|} \times \sum_{p \in S_C} p$

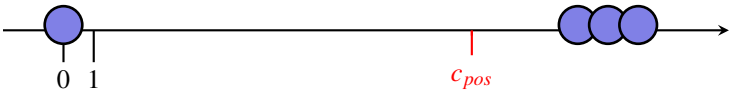
Although CoGiL is extremely similar to CoG, proving its correctness is not. For example, Cohen and Peleg [10] first used in the conference version of their paper moments of inertia as a monotonically decreasing measure to prove the convergence of CoG:

$$I(q) = \frac{1}{|C|} \times \sum_{r \in C} \|C(r) - q\|^2$$

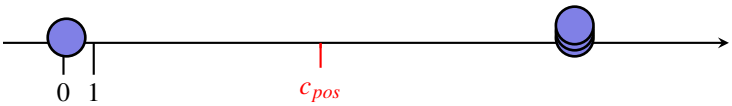
Expressing this measure with the observed spectrum gives:

$$I(q) = \frac{1}{|S_C|} \times \sum_{p \in S_C} \|p - q\|^2$$

Now, without strong global multiplicity detection, it is possible that this measure does *not* decrease monotonically for c_{pos}. For example, consider four robots in a one-dimension metric space, localised at locations $0; 17; 18; 19$.



The centre of gravity of the inhabited locations c_{pos} is at 13.5 and $I(c_{pos}) = 61.25$. Now, consider that $\delta = 0.1$. A possible following configuration is that the robot at 0 has moved by δ toward c_{pos} and the others have stopped at location 16.9.



The centre of gravity of the inhabited locations c_{pos} is now at 8.5, and $I(c_{pos}) = 70.56$, which is strictly greater than its previous value. So, the proof argument appearing in Cohen and Peleg's conference paper [10] does not extend to the case without global strong multiplicity detection.

Fortunately, the underlying idea of the proof appearing in the journal version of Cohen and Peleg [11] can be extended to the case without multiplicity detection. We thus construct our certified proof along the main arguments of theirs.

3.2 Formalisation, and key points to prove correctness

Gathering in the context of flexible movements. A way to state Gathering and Convergence has been already described in [2, 13]. Those definitions take place in a context where movements are rigid, and thus the specification of what a solution to Gathering is has to be generalised for the case of flexible movements. We name `gathered_at` *pt* the property of a configuration the robots of which are all gathered at the same location *pt*.

```
Definition gathered_at (pt : Loc.t) (conf : Config.t) :=
  ∀ g : G, Loc.eq (conf (Good g)) pt.
```

We say that a location *pt* and an execution enjoy the property `Gather` if all robots are gathered at *pt* for all rounds of the (infinite) execution.

```
Definition Gather (pt: Loc.t) (e : execution) : Prop :=
 Streams.forever (Streams.instant (gathered_at pt)) e.
```

`WillGather` *pt e* means that the (infinite) execution *e* is *eventually* `Gather`ed for *pt*. That is: there is a (finitely) reachable instant in *e* for which *pt* and what remains of *e* fulfill `Gather`.

```
Definition WillGather (pt : Loc.t) (e : execution) : Prop :=
 Streams.eventually (Gather pt) e.
```

We may now characterise that a robogram *r* is a solution to the Gathering problem for a demon *d*, in the context of δ-flexible movements. It takes into account the minimal distance of travel δ that is necessary to define the execution (the final theorem will be based on this characterisation, for any fully synchronous demon *d* and any strictly positive distance δ).

```
Definition FullSolGathering (r : robogram) (d : demon) δ :=
  ∀ config, ∃ pt : Loc.t, WillGather pt (execute δ r d config).
```

Expressing the protocol in Pactole. The space of locations is $\mathbb{R}^2$ and its type is `R2.t` in the following. Writing the algorithm is straightforward in our framework, and the Coq implementation is almost exactly an actual robot code. Let `ffgatherR2_pgm` denote the code of the algorithm, which takes a spectrum as an input and returns a location, and let `ffgatherR2` denote the robogram, that is the code and its property of invariance through equivalent spectra.

```
Definition ffgatherR2_pgm (s : Spect.t) : R2.t :=
  let spect := Spect.elements s in
    match spect with
      | nil ⇒ (0, 0)                                    (* no robot *)
      | pt :: nil ⇒ pt                                  (* gathered *)
      | _ :: _ :: _ ⇒ barycenter spect
    end.
```

The function computing the barycentre is simply:

```
Definition barycenter (E: list R2.t) : R2.t :=
 1 / (INR (List.length E)) * (List.fold_left R2.add E R2.origin).
```

where `INR` injects a natural number into reals.

The robogram can be expressed in the demon's frame of reference. The input spectrum given to the code above is expressed in the robot's frame of reference (it is a local code). As noticed in [14], we establish explicitly and formally that it is sufficient to reason about the protocol in the frame of reference of the demon. The geometrical concepts in use in the protocol are invariant under the changes of frame that are allowed: scaling, rotation, symmetry and translation, hence we can express the global configuration after one round without making reference to the frames of each robot (lemma `round_simplify`).

```
Theorem round_simplify : ∀ da conf δ,
  Config.eq (round δ ffgatherR2 da conf)
    (fun id ⇒ match da.(step) id with
              | None ⇒ conf id
              | Some (f, r) ⇒
                let s := Spect.from_config conf in
                match Spect.elements s with
                | nil ⇒ conf id (* only happen with no robots *)
                | pt :: nil ⇒ pt (* done *)
                | _ ⇒ let move := (r * (barycenter (Spect.elements s)
                                        - (conf id)))ℝ² in
                      if Rle_bool δ (R2norm move)
                      then ((conf id) + move)ℝ²
                      else barycenter (Spect.elements s)
                end
              end).
```

Eventually no-one moves. The main difficulty is to establish that after a finite number of steps, no robot will change its location. This amounts to finding a measure that decreases for a well founded ordering along the execution.

To this goal, we consider the maximal distance $dm(C)$ between any two robots in a configuration C.

```
Definition measure (conf: Config.t) : ℝ :=
 max_dist_spect (Spect.from_config conf).
```

where `max_dist_spect` is computed by two nested recursions (`fold_right`) over the list of inhabited locations (`Spect.elements spect`):

```
(* Maximal distance from a point to a list of points. *)
Definition max_dist_R2_pt_list (pt: R2.t) (l: list R2.t) : ℝ :=
  fold_right (fun pt1 max → Rmax (R2.dist pt pt1) max) 0 l.

(* Maximal distance between points of two lists. *)
Definition max_dist_R2_list_list (l1: list R2.t) (l2: list R2.t): ℝ :=
  fold_right (fun pt0 max → Rmax max (max_dist_R2_pt_list pt0 l2))
             0 l1.

(* Maximal distance between two points in a spectrum. *)
Definition max_dist_spect (spect: Spect.t) : ℝ :=
  max_dist_R2_list_list (Spect.elements spect)
                        (Spect.elements spect).
```

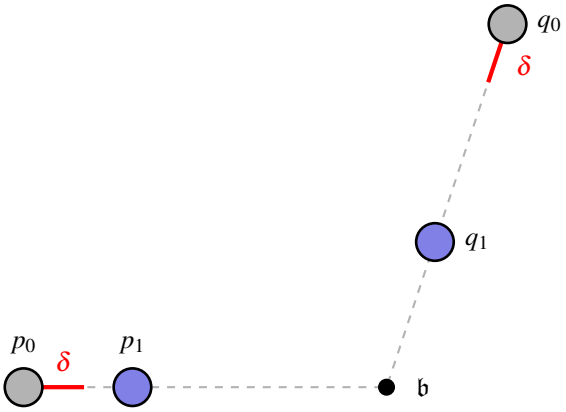
If this distance is less than δ then after one step all robots are gathered and we are done, as stated by Theorem `round_last_step`:

```
Theorem round_last_step: ∀ d conf δ,
  δ > 0 →
  FullySynchronous d →
  measure conf ≤ δ →
  measure (round δ ffgatherR2 (head d) conf) = 0.
```

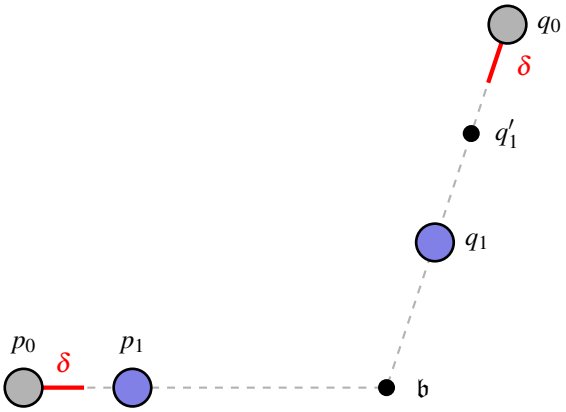
If this distance is not less than δ, we prove that if a configuration C_1 is obtained after one round from a configuration C_0 such that $dm(C_0) > \delta$, then $dm(C_1) \leq dm(C_0) - \delta$. This part is established through Theorem `round_lt_config`:

```
Theorem round_lt_config: ∀ d conf δ,
  δ > 0 →
  FullySynchronous d →
  δ < measure conf →
  measure (round δ ffgatherR2 (head d) conf) ≤ measure conf - δ.
```

The crucial step in establishing this lemma is to prove that for any two inhabited locations p_1 and q_1 in C_1, $\|p_1 - q_1\| \leq dm(C_0) - \delta$. Let us denote by $\mathfrak{b}$ the location of the barycentre of inhabited locations in C_0. As locations p_1 and q_1 are inhabited in C_1, we can assume that some robots P and Q occupying them in C_1 were previously in C_0 at respectively p_0 and q_0.



Now let us perform a case analysis on whether $\|p_0 - \mathfrak{b}\|$ and $\|q_0 - \mathfrak{b}\|$ are greater or equal to δ; the only interesting case is the non-degenerate one where both are greater. In this case, P and Q move towards $\mathfrak{b}$, and in particular $p_1 = p_0 + \kappa \times (\mathfrak{b} - p_0)$ and $q_1 = q_0 + \mu \times (\mathfrak{b} - q_0)$ for $\kappa, \mu \in [0,1]$. Let us suppose $\kappa \leq \mu$ (the other case is symmetrical) and write q_1' the point given by $q_1' = q_0 + \kappa \times (\mathfrak{b} - q_0)$.



By Thales's Basic Proportionality Theorem we have

$$\|p_1 - q_1'\| = (1 - \kappa) \times \|p_0 - q_0\| \tag{1}$$

and thus

$$\|p_1 - q_1'\| \leq (1-\kappa) \times dm(C_0) \tag{2}$$

To conclude we need two lemmas. One states that the distance from any robot to the barycentre of locations $\mathfrak{b}$ is less than or equal to $dm(C_0)$:

```
Lemma barycenter_dist_decrease:
  ∀ (E: list R2.t) (dm: R) (c: R2.t),
    E ≠ nil →
    (∀ p1 p2, In p1 E → In p2 E → R2.dist p1 p2 ≤ dm) →
    c = barycenter E →
    ∀ p, In p E → R2.dist p c ≤ dm.
```

where the predicate `In p E` means that the point `p` is in the list `E`. Using this lemma we have

$$\|p_0 - \mathfrak{b}\| \leq dm(C_0) \tag{3}$$

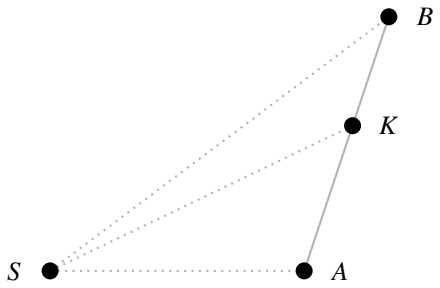
from which we deduce

$$\|p_1 - \mathfrak{b}\| \leq (1-\kappa) \times dm(C_0) \tag{4}$$

The other lemma states that the length of a segment contained in a triangle is smaller than the length of at least one of the sides of the triangle:

```
Lemma inner_segment :
  ∀ A B S K,
  not R2.eq A B →
  on_segment A B K →
  R2.dist S K ≤ R2.dist S A  ∨  R2.dist S K ≤ R2.dist S B.
```

where `on_segment A B K` means that the point `K` is between `A` and `B`.



From this lemma we have either

$$\|p_1 - q_1\| \leq \|p_1 - q_1'\| \tag{5}$$

or

$$\|p_1 - q_1\| \leq \|p_1 - \mathfrak{b}\| \tag{6}$$

Then using equation 2 or 4 we conclude

$$\|p_1 - q_1\| \leq (1-\kappa) \times dm(C_0) \tag{7}$$

and finally

$$\|p_1 - q_1\| \leq dm(C_0) - \delta \tag{8}$$

which concludes the proof of `round_lt_config`. □

Finally, the lemmas `round_last_step` and `round_lt_config` being proved, we may then take as a relevant indication for a configuration C the natural number $m(C) = \lceil \frac{dm(C)}{\delta} \rceil$ and define accordingly the ordering we use:

```
Definition lt_config δ x y :=
 (ℤ.to_nat (up(measure x / δ))) < (ℤ.to_nat (up(measure y / δ))).
```

which is well-founded over the naturals.

Robots stay gathered forever. As there is only one phase in the algorithm, the computed target is always the barycentre of the inhabited locations, which is the same for all robots. We need however technical lemmas to complete the final proof. Firstly that when robots are gathered, they will stay forever at the same location, namely:

```
Lemma gathered_at_OK : ∀ δ d conf pt, gathered_at pt conf
→ Gather pt (execute δ ffgatherR2 d conf).
```

The counterpart is that a robot that is not at the barycentre of inhabited locations will actually move (that is, it will change its location).

```
Lemma not_barycenter_moves: ∀ δ d conf gid, δ > 0
→ FullySynchronous d
→ ¬ R2.eq (conf gid) (barycenter (Spect.elements (!! conf)))
→ ¬ R2.eq (round δffgatherR2 (Streams.hd d) conf gid) (conf gid).
```

We are now ready to tackle the final proof.

The final theorem states that for all positive δ, the robogram `ffgatherR2` is a solution to the gathering problem in FSYNC.

```
 Theorem FSGathering_in_R2 : ∀ δ d, δ > 0
→ FullySynchronous d
→ FullSolGathering ffgatherR2 d δ.
```

It is proven via well-founded induction over `lt_config` and by case analysis: if the robots are already gathered or will be gathered at the next step then we are done, else we use `round_lt_config`. That last proof is about 20 lines of Coq.

3.3 Certifying Cohen and Peleg's Gathering Algorithm

As the informal proof argument is similar between Cohen and Peleg's algorithm [10, 11] (that makes use of multiplicity detection) and ours, it is worth investigating whether their certified proof arguments can be somewhat reused. In this section, we report how minor changes in the proof of our CoGIL algorithm permit to obtain a formal certified proof of the CoG FSYNC gathering algorithm by Cohen and Peleg [10, 11].

The core difference between the two approaches lies in the definition of the spectrum (that is a multiset in the Cohen and Peleg's approach, while ours uses a simple set), as the algorithm remains the same (taking the spectrum as input). Only few technical changes (*e.g.*, with respect to the properties that are associated to the spectrum) are necessary before the new proof is obtained.

More specifically, the instanciation of the spectrum for the CoGIL algorithm was:

```
Module Spect := SetSpectrum.Make(Loc)(N)(Names)(Info)(Config).
```

For Cohen and Peleg's CoG algorithm, the instanciation simply becomes:

```
Module Spect := MultisetSpectrum.Make(Loc)(N)(Names)(Info)(Config).
```

Note that this is the only noticeable change in the definitions.

4 Discussion and Perspectives

We presented the first FSYNC gathering protocol, CoGiL, that does not require robots to be capable of multiplicity detection (nor any other extra assumptions), closing the only remaining open case in Prencipe's set of impossibility results [19]. We advocate that proofs for even small variants of oblivious mobile robot protocols (such a CoGiL, which is a minor variant of Cohen and Peleg's CoG protocol) should be thoroughly checked from the beginning, using mechanised support such as a proof assistant. This methodology enabled the possibility to present a proof for our protocol, whose correctness can be certified. Our framework also permitted to certify the correctness of Cohen and Peleg's solution that uses multiplicity detection, demonstrating the versatility of our approach and its ability to cope with various assumptions and model hypotheses.

We want to stress that, even if the actual development of a formal proof remains a difficult task, the *specifications* of properties and protocols in our framework do not require a strong expertise with the Coq proof assistant. As an illustration, many of the specifications appearing in this paper, most notably the specification of the actual protocol, were developed by one of the authors while a M1-level trainee (first year master, Bologna process).

Whith respect to the number of lines of code in the Coq development, we distinguish the specification part and the proof part. When the problem and space domain (this paper considers an Euclidean continuous space, yet recent advances [4] provide Coq foundations for robots evolving in discrete spaces, *i.e.* graphs) are already defined in our framework, a user in charge of specifying may only have to provide the code of the algorithm itself, along with the statements of its invariance through equivalent spectra, the statement of the main theorem (that is, the correctness of the protocol), which is rather short. Being very parametric, our framework simplifies nonetheless the writing of extensions (*i.e.*, specification of new properties) and our current effort is targeted toward including as many classical notions and variants that appear in the literature, facilitating the specification of new problems and protocols as much as possible.

We believe a thorough revision of other published results in the context of oblivious mobile robots will lay a solid foundation for further research advances. Thanks to the collaborative effort of the Pactole framework, reuse of previous achievements is facilitated and encouraged.

Acknowledgements

The authors are grateful to the reviewers who provided constructive comments and helped to improve the presentation of this work.

References

1. Karine Altisen, Pierre Corbineau, and Stéphane Devismes. A framework for certified self-stabilization. In Elvira Albert and Ivan Lanese, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 36th IFIP WG 6.1 International Conference, FORTE 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings*, volume 9688 of *Lecture Notes in Computer Science*, pages 36–51. Springer-Verlag, 2016.
2. Cédric Auger, Zohir Bouzid, Pierre Courtieu, Sébastien Tixeuil, and Xavier Urbain. Certified Impossibility Results for Byzantine-Tolerant Mobile Robots. In Teruo Higashino, Yoshiaki Katayama, Toshimitsu Masuzawa, Maria Potop-Butucaru, and Masafumi Yamashita, editors, *Stabilization, Safety, and Security of Distributed Systems - 15th International Symposium (SSS 2013)*, volume 8255 of *Lecture Notes in Computer Science*, pages 178–186, Osaka, Japan, November 2013. Springer-Verlag.
3. Thibaut Balabonski, Pierre Courtieu, Lionel Rieg, Sébastien Tixeuil, and Xavier Urbain. Certified gathering of oblivious mobile robots: Survey of recent results and open problems. In Laure Petrucci, Cristina Seceleanu, and Ana Cavalcanti, editors, *Critical Systems: Formal Methods and Automated Verification - Joint 22nd International Workshop on Formal Methods for Industrial Critical Systems - and - 17th International Workshop on Automated Verification of Critical Systems, (FMICS-AVoCS 2017)*, volume 10471 of *Lecture Notes in Computer Science*, pages 165–181, Turin, Italy, September 2017. Springer-Verlag.
4. Thibaut Balabonski, Robin Pelle, Lionel Rieg, and Sébastien Tixeuil. A foundational framework for certified impossibility results with mobile robots on graphs. In *Proceedings of International Conference on Distributed Computing and Networking*, Varanasi, India, January 2018.
5. Béatrice Bérard, Pascal Lafourcade, Laure Millet, Maria Potop-Butucaru, Yann Thierry-Mieg, and Sébastien Tixeuil. Formal verification of mobile robot protocols. *Distributed Computing*, 29(6):459–487, 2016.
6. Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer-Verlag, 2004.
7. François Bonnet, Xavier Défago, Franck Petit, Maria Potop-Butucaru, and Sébastien Tixeuil. Discovering and assessing fine-grained metrics in robot networks protocols. In *33rd IEEE International Symposium on Reliable Distributed Systems Workshops, SRDS Workshops 2014, Nara, Japan, October 6-9, 2014*, pages 50–59. IEEE, 2014.
8. Béatrice Bérard, Pierre Courtieu, Laure Millet, Maria Potop-Butucaru, Lionel Rieg, Nathalie Sznajder, Sébastien Tixeuil, and Xavier Urbain. Formal Methods for Mobile Robots: Current Results and Open Problems. *International Journal of Informatics Society*, 7(3):101–114, 2015. Invited Paper.
9. Pierre Castéran and Vincent Filou. Tasks, types and tactics for local computation systems. *Studia Informatica Universalis*, 9(1):39–86, 2011.
10. Reuven Cohen and David Peleg. Robot Convergence via Center-of-Gravity Algorithms. In Rastislav Kralovic and Ondrej Sýkora, editors, *Structural Information and Communication Complexity - 11th International Colloquium (SIROCCO 2004)*, volume 3104 of *Lecture Notes in Computer Science*, pages 79–88, Smolenice Castle, Slowakia, June 2004. Springer-Verlag.
11. Reuven Cohen and David Peleg. Convergence Properties of the Gravitational Algorithm in Asynchronous Robot Systems. *SIAM Journal of Computing*, 34(6):1516–1528, 2005.
12. Thierry Coquand and Christine Paulin-Mohring. Inductively Defined Types. In Per Martin-Löf and Grigori Mints, editors, *International Conference on Computer Logic (Colog'88)*, volume 417 of *Lecture Notes in Computer Science*, pages 50–66. Springer-Verlag, 1990.
13. Pierre Courtieu, Lionel Rieg, Sébastien Tixeuil, and Xavier Urbain. Impossibility of Gathering, a Certification. *Information Processing Letters*, 115:447–452, 2015.

14. Pierre Courtieu, Lionel Rieg, Sébastien Tixeuil, and Xavier Urbain. Certified universal gathering algorithm in $\mathbb{R}^2$ for oblivious mobile robots. In Cyril Gavoille and David Ilcinkas, editors, *Distributed Computing - 30th International Symposium, (DISC 2016)*, volume 9888 of *Lecture Notes in Computer Science*, Paris, France, September 2016. Springer-Verlag.
15. Stéphane Devismes, Anissa Lamani, Franck Petit, Pascal Raymond, and Sébastien Tixeuil. Optimal Grid Exploration by Asynchronous Oblivious Robots. In Andréa W. Richa and Christian Scheideler, editors, *Stabilization, Safety, and Security of Distributed Systems - 14th International Symposium (SSS 2012)*, volume 7596 of *Lecture Notes in Computer Science*, pages 64–76, Toronto, Canada, October 2012. Springer-Verlag.
16. Ha Thi Thu Doan, François Bonnet, and Kazuhiro Ogata. Model checking of robot gathering. In James Aspnes and Pascal Felber, editors, *Principles of Distributed Systems - 21th International Conference (OPODIS 2017)*, Leibniz International Proceedings in Informatics (LIPIcs), Lisbon, Portugal, December 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
17. Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012.
18. Laure Millet, Maria Potop-Butucaru, Nathalie Sznajder, and Sébastien Tixeuil. On the synthesis of mobile robots algorithms: The case of ring gathering. In Pascal Felber and Vijay K. Garg, editors, *Stabilization, Safety, and Security of Distributed Systems - 16th International Symposium, (SSS 2014)*, volume 8756 of *Lecture Notes in Computer Science*, pages 237–251, Paderborn, Germany, September 2014. Springer-Verlag.
19. Giuseppe Prencipe. Impossibility of gathering by a set of autonomous mobile robots. *Theoretical Computer Science*, 384(2-3):222–231, 2007.
20. Sasha Rubin, Florian Zuleger, Aniello Murano, and Benjamin Aminof. Verification of asynchronous mobile-robots in partially-known environments. In Qingliang Chen, Paolo Torroni, Serena Villata, Jane Yung-jen Hsu, and Andrea Omicini, editors, *PRIMA 2015: Principles and Practice of Multi-Agent Systems - 18th International Conference, Bertinoro, Italy, October 26-30, 2015, Proceedings*, volume 9387 of *Lecture Notes in Computer Science*, pages 185–200. Springer-Verlag, 2015.
21. Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2012.
22. Arnaud Sangnier, Nathalie Sznajder, Maria Potop-Butucaru, and Sébastien Tixeuil. Parameterized verification of algorithms for oblivious robots on a ring. In *Formal Methods in Computer Aided Design*, Vienna, Austria, October 2017.
23. Ichiro Suzuki and Masafumi Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *SIAM Journal of Computing*, 28(4):1347–1363, 1999.

A Axioms of the formalisation

In the main file `Gathering/InR2/FSyncFlexNoMultAlgorithm.v`, the last command: `Print Assumptions Gathering_in_R2` shows all the axioms upon which the proof of correctness of our algorithm for gathering in $\mathbb{R}^2$ relies, in total 31 axioms. Here, we break them down. They can be classified in three categories:

- The first category is the axiomatisation of reals numbers from the Coq standard library. It represents by far the biggest number of axioms (26), and they are not listed here.
- The second category is the description of the problem.

  ```
  nG : nat
  Hyp_nG : 2 ≤ nG
  ```

  As one can see, it simply means that our proof is valid for any number `nG` of robots greater than or equal to 2. Notice that with one robot or less, the problem is not interesting (trivially solved).
- The third category contains three usual geometric properties that are not part of our library. These three axioms are the only ones which could be seen as real axioms to be proved, the previous two categories being the parameters of the problem. On the one hand, there are some properties about barycentres that we think could be provable from its axiomatisation but are currently left as axioms: that the barycentre is unique and that the result of the function computing the barycentre is indeed a barycentre:

  ```
  barycenter_n_unique : ∀ (E : list R2.t) (a b : R2.t),
   is_barycenter_n E a → is_barycenter_n E b → R2.eq a b

  barycenter_n_spec : ∀ E : list R2.t,
   is_barycenter_n E (barycenter E)
  ```

  On the other hand, there is the proof that similarities can be expressed with an orthogonal matrix M, a zoom factor λ and a translation t: for any similarity s, we can find $M \in \mathscr{O}_2(\mathbb{R})$, $\lambda \in \mathbb{R}^+$ and $t \in \mathbb{R}^2$ such that $s = \lambda M + t$. For convenience, the orthogonal matrix and the zoom factor are combined into two column vectors u and v: we have $\lambda M = (\mathrm{u}\ \mathrm{v})$ with $u \perp v$ and $\|u\| = \|v\| = \lambda$.

  ```
  similarity_in_R2 : ∀ sim : Sim.t, ∃ u v t : R2.t,
     R2norm u = Sim.zoom sim
  ∧  R2norm v = Sim.zoom sim
  ∧  perpendicular u v
  ∧  (∀ pt : R2.t,
       sim pt = (product u pt * u + product v pt * v + t)_ℝ²)
  ```

  These types of axioms can be discharged through the connection with Coq libraries dedicated to geometry.