

Certified Impossibility Results for Byzantine-Tolerant Mobile Robots

Cédric Auger, Zohir Bouzid, Pierre Courtieu, Sébastien Tixeuil, Xavier Urbain

▶ **To cite this version:**

Cédric Auger, Zohir Bouzid, Pierre Courtieu, Sébastien Tixeuil, Xavier Urbain. Certified Impossibility Results for Byzantine-Tolerant Mobile Robots. [Research Report] 1560, LRI - CNRS, University Paris-Sud. 2013. hal-00834633

HAL Id: hal-00834633
https://hal.science/hal-00834633

Submitted on 18 Jun 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

RAPPORT DE RECHERCHE

L R I

CERTIFIED IMPOSSIBILITY RESULTS FOR BYZANTINE-TOLERANT MOBILE ROBOTS

AUGER C / BOUZID Z / COURTIEU P / TIXEUIL S / URBAIN X

Unité Mixte de Recherche 8623
CNRS-Université Paris Sud -LRI

06/2013

Rapport de Recherche N° 1560

CNRS - Université de Paris Sud
Centre d'Orsay
LABORATOIRE DE RECHERCHE EN INFORMATIQUE
Bâtiment 650
91405 ORSAY Cedex (France)

Certified Impossibility Results for Byzantine-Tolerant Mobile Robots*

Cédric Auger, Zohir Bouzid[4], Pierre Courtieu[2],
Sébastien Tixeuil[4,5], and Xavier Urbain[1,3]

[1] École Nat. Sup. d'Informatique pour l'Industrie et l'Entreprise (ENSIIE), Évry, F-91025
[2] CÉDRIC – Conservatoire national des arts et métiers, Paris, F-75141
[3] LRI, CNRS UMR 8623, Université Paris-Sud, Orsay, F-91405
[4] UPMC Sorbonne Universités
[5] Institut Universitaire de France

Abstract. We propose a framework to build formal developments for robot networks using the COQ proof assistant, to state and to prove formally various properties. We focus in this paper on *impossibility* proofs, as it is natural to take advantage of the COQ higher order calculus to reason about algorithms as abstract objects. We present in particular formal proofs of two impossibility results for convergence of oblivious mobile robots if respectively more than one half and more than one third of the robots exhibit Byzantine failures, starting from the original theorems by Bouzid *et al.*. Thanks to our formalization, the corresponding COQ developments are quite compact. To our knowledge, these are the first certified (in the sense of formally proved) impossibility results for robot networks.

* This work was supported in part by the Digiteo Île-de-France project PACTOLE 2009-38HD.

1 Introduction

Networks of static and/or mobile sensors (that is, robots) [17] received increasing attention in the past few years from the Distributed Computing community. On the one hand, the use of cooperative swarms of inexpensive robots to achieve various complex tasks in potentially hasardous environments is a promising option to reduce human and material costs and assess the relevance of Distributed Computing in a practical setting. On the other hand, execution model differences warrant extreme care when revisiting "classical results" from Distributed Computing, as very small changes in assumed hypotheses may completely change the feasibility of a particular problem. Negative results such as impossibility results are fundamental in Distributed Computing to establish what can and cannot be computed in a given setting, or permitting to assess optimality results through lower bounds for given problems. Two notorious examples are the impossibility of reaching consensus in an asynchronous setting when a single process may fail by stopping unexpectedly [16], and the impossibility of reliably exchanging information when more than one third of the processes can exhibit arbitrary behaviour [27]. As noted by Lamport [23], correctly proving results in the context of Byzantine (*a.k.a.* arbitrary behaviour capable) processes is a major challenge, as [they knew] *of no area in computer science or mathematics in which informal reasoning is more likely to lead to errors than in the study of this type of algorithm*.

An attractive way to assess the validity of distributed algorithm is to use *tool assisted* verification, be it based process algebra [3, 18], local computations [25], `Event-B` [7], Coq [8], HOL [9], Isabelle/HOL [21], or TLA [23, 22] that can enjoy an Isabelle back-end for its provers [12]. Surprisingly, only few works consider using mechanized assistance for networks of mobile entities, be it population protocols [13, 10] or mobile robots [14, 4]. In this paper, our goal is to propose a formal provable framework in order to prove positive or negative results for localised distributed protocols in mobile robotic networks, based on recent advances in mechanical proving and related areas, and in particular on *proof assistants*. Proof assistants are environments in which a user can express programs, state theorems and develop interactively proofs that will be mechanically checked (that is machine-checked). They have been successfully employed for various tasks such as the formalisation of programming language semantics [24, 26], verification of cryptographic protocols [2], certification of RSA keys [29], mathematical developments as involved as the 4-colours [19] or Feit-Thompson [20] theorems.

Our contribution We developed a general framework relying on the COQ proof assistant to prove possibility and impossibility results about mobile robotic networks. The key property of our approach is that its underlying calculus is of higher order: instead of providing the code of the distributed protocols executed by the robots, we may quantify universally on those programs/algorithms, or just characterize them with an abstract property. This genericity makes this approach complementary to the use of model-checking methods for verifying distributed algorithms [6, 10, 14] that are highly automatic, but address mainly particular instances of algorithms. In particular, quantifying over algorithms allows us to express in a natural way *impossibility results*.

We illustrate how our framework allows such certification by providing COQ proofs of two earlier impossibility and lower bound theorems by Bouzid *et al.* [5], guaranteeing soundness of the first one, and of the SSYNC fair version of the second one. More precisely, in the context[1] of oblivious robots that are endowed with strong global multiplicity detection and whose movements are constrained along a rational line, and assuming that the demon (that is, the way robots are scheduled for execution) is fair, the convergence problem cannot be solved if respectively not less than one half (Theorem 1) and not less than one third (Theorem 2) of robots are Byzantine.

The interestingly short size of the COQ proofs we obtained using our framework not only makes it easily human-readable, but is also very encouraging for future applications and extensions of our framework.

Related work. With reference to proof assistants, Küfner *et al.* [21] develop a methodology to develop ISABELLE-checked proofs of properties of fault-tolerant distributed algorithms in a asynchronous message passing style setting. This work's motivations are similar to ours, however the setting (message passing distributed algorithms) is different, moreover it focuses on positive results only whereas we provide negative results, *i.e.* proofs of impossibility.

Chou [9] develops a methodology based on the HOL proof assistant to prove properties of concrete distributed algorithms via proving simulation with abstract ones. The methodology does not allow to prove impossibility results. Casteran *et al.* [8] propose proofs of negatives results in COQ for some kinds of distributed algorithms. Though very interesting, their approach is based on labeled graph rewriting and does not address robot networks. Another interesting approach is that of Deng and Monin [13] that uses COQ to prove the correctness of distributed self-stabilizing protocols in the population protocol model. This model permits to describe interactions of an arbitrary large size of mobile entities, but the considered entities lack movement control and geometric awareness

[1] Distributed Robot model assumptions are presented in Section 2.

that are characteristic of robot networks such as those we envision, and is thus not suitable for our purpose. This approach also only considers positive results.

Preliminary attempts for automatically proving impossibility results in robot networks properties are due to Devismes *et al.* [14] and to Bonnet *et al.* [4]. The first paper uses LUSTRE formalism and model-checking to search exhaustively all possible 3-robots protocols that explore every node of a 3×3 grid (and conclude that no such algorithm exists). The second paper uses an ad hoc tool to generate all possible unambiguous protocols of k robots operating in an n-sized ring (k and n are given as parameters) and check exhaustively the properties of the generated protocols (and in the paper conclude that no protocol of 5 robots on a 10 sized ring can explore all nodes infinitely often with every robot). Those two proposals differ from our goal in several ways. Firstly, they are limited to a so called *discrete space*, where the robots may only occupy a *finite* number of positions, while we focus on the more realistic setting where an infinite number of positions are possible for the robots. Also, contrary to both, we do not want to restrict our tools to a particular setting (*e.g.* 3 robots on a 3×3 grid), but rather have results that are general with respect to all considered parameters. Then, unlike the second proposal, we want universal impossibility results (*i.e.* consider not only unambiguous protocols – that permit to limit combinatorial explosion to some extend – but also ambiguous ones – resulting from symmetrical situations that are likely to occur in practice). Finally, we want to integrate the possibility of misbehaving robots (*e.g.* robots crashing or exhibiting arbitrary and potentially malicious behaviour), rather than assuming that all considered robots are correct. This enables to state formally and assess the amount of faults and attack resilience a given robot protocol may guarantee, which is crucial when robots are deployed in dangerous areas as it is often the case.

Roadmap. The sequel of the paper is organized as follows. First, we recall the context of robot networks in Section 2. Then, in Section 3 we give a brief description of Coq and its main principles. Section 4 contains the basis of our formal model for robot networks, and some useful theorems. We show in Section 5 how convenient it is to carry out formal proofs of various properties, as we study previous results by Bouzid *et al.* [5]. We provide some concluding remarks in Section 6.

Note that for the sake of readability we slightly simplified Coq notations (mostly to avoid syntactic sugar). The actual development for Coq 8.4pl3 is available at `http://pactole.lri.fr/`

2 Robot Networks

We borrow most of the notions in this section from [28, 1, 17]. The network consists in a set of n mobile entities, called robots, arbitrarily located in the space. Robots cannot communicate directly by sending messages to each others. Instead, their communication is based on vision: they observe the positions of other robots, and based on their observations, they compute destination points to which they move.

Robots are *homogeneous* and *anonymous*: they run the same algorithm (called *robogram*), they are completely indistinguishable by their appearance, and no identifier can be used in their computations. They are also *oblivious*, i.e. they cannot remember any previous observation, computation or movement performed in any previous step.

For simplicity, we assume that robots are *without volume*, *i.e.* they are modeled as points that cannot obstruct the movement or vision of other robots. Visibility is *global*: the entire set of robots can always be seen by any robot at any time. Robots that are able to determine the exact number of robots occupying a same position enjoy *strong* multiplicity detection ; if they can only know if a given position is inhabited or not, their multiplicity detection is said to be *weak*. Each robot has its own local coordinate system and its own unit measure. They do not share any origin, orientation, and more generally any frame of reference.

The multiset of positions of robots at a given time is called a *configuration*. We assume that the actions of robots are controlled by a fictitious entity called the *demon* (or adversary). Each time a robot is activated by the demon, it executes a complete three-phases cycle: Look, Compute and Move. During the Look phase, using its visual sensors, the robot gets a snapshot of the current configuration. Then, based only on this observed configuration, it computes a destination in the Compute phase using its robogram and moves towards it during the subsequent Move phase. Movements of robots are *atomic*, *i.e.* the demon cannot stop them before they reach the destination.

A *run* (or execution) is an infinite sequence of rounds. During each round, the demon chooses a subset of robots and activates them to execute a cycle. We assume the scheduling to be *fair*, *i.e.* each robot is activated infinitely often in any infinite execution, and *atomic* in the sense that robots that are activated at the same round execute their actions synchronously and atomically. An atomic demon is called fully-synchronous (FSYNC) if all robots are activated at each round, otherwise it is said to be semi-synchronous (SSYNC). The impossibility results we focus on are given in the FSYNC and SSYNC models, and hence remain valid in less constrained ones (*e.g.* non-atomic, unfair scheduling, etc.).

A robot is *Byzantine* (or faulty) if it does not comply with the robogram and behaves in arbitrary and unpredictable way. We assume that the movements of Byzantine robots are controlled by the adversary that uses them in order to make the algorithm fail. Let $f \in [0, n]$ be a parameter that denotes the number of faulty robots. Robots that are not Byzantine are called *correct*. Correct robots are supposed to know an upper bound on the number of Byzantine robots.

3 The COQ Proof Assistant

COQ is based on *type theory*. Its *formal language* can express objects, properties and proofs in a unified way; all these are represented as terms of an expressive λ-calculus: the *Calculus of Inductive Constructions* (CIC) [11]. λ-abstraction is denoted `fun x:T ⇒ t`, and application is denoted `t u`. A proof development with COQ consists in trying to build, interactively and using tactics, a λ-term the type of which corresponds to the proven theorem (Curry-Howard style).

The kernel of COQ is a *proof checker* which checks the validity of proofs written as CIC-terms. Indeed, in this framework, a term is a *proof* of its type, and checking a proof consists in typing a term. Roughly speaking, the small kernel of COQ simply type-checks λ-terms to ensure soundness.

A very powerful feature of COQ is the ability to define *inductive types* to express inductive data types and inductive properties. For example the following inductive types define the data type `nat` of natural numbers, `O` and `S` (successor) being the two constructors, and the property `even` of being an even natural number. In this setting the term `even_S(S(S O))(even_S O (even_O))` is of type `even(S(S(S(S O))))` so it is a proof that 4 is even.

```
Inductive nat : Set := O : nat | S : nat → nat.
Inductive even : nat → Prop :=
  | even_O : even O
  | even_S : ∀ n : nat, even n → even (S(S n)).
```

We also make use of *coinductive* types to express infinite data types and properties on them. For example in the robot networks setting a set of robots has an infinite behaviour. For example one can define infinite streams of natural numbers and the property `all_even` of being a infinite stream of even natural number as follows:

```
CoInductive stm : Set :=
  | scons : nat → stm → stm.
CoInductive all_even : stm → Prop :=
  | Ceven_all: ∀ n s, even n → all_even s → all_even (scons n s).
```

4 The formal model

We present our formal model and the relevant notations. Robots are anonymous, however we need to identify some of them in the proofs. Thus, we consider the union of two given disjoint finite sets of *identifiers*: G referring to robots that behave correctly, and B referring to the set of Byzantine ones[2]. Note that those sets are isomorphic to segments of $\mathbb{N}$ but we keep our formalisation as abstract as possible. If needed in the model, we can make sure that names are not used by the embedded algorithm, as shown below.

```
Variable G B : finite.
Inductive ident := Good : G → ident | Byz : B → ident.
```

Locations, Positions, Similarities. Robots are distributed in space, at places called *locations*. We define a *position* as a *function* from a set of identifiers to the space of locations. As the space of locations in the paper of Bouzid *et al.* [5] is an infinite line, we use $\mathbb{Q}$ for locations. Note that going from one to many dimensions is not a problem with respect to our formalisation. Throughout this article, and unless specified otherwise `gp` denotes a position for correct robots, and `bp` a position for Byzantine ones. The position of all robots is then given by the combination $\texttt{gp} \uplus \texttt{bp}$.

```
Record position:= { gp: G → location ; bp: B → location }.
(* Getting the location of a robot *)
Definition locate p (id: ident): location :=
  match id with
  | Good g ⇒ p.(gp) g
  | Byz b ⇒ p.(bp) b end.
```

Robots compute their target position from the observed configuration of their siblings in the considered space. We also define permutations of robots, that is bijective applications from $G \cup B$ to itself, usually denoted hereafter by Greek letters. Moreover, any correct robot is supposed to act as any other correct robot in the same context, that is, with a *similar* perception of the environment. For two rational numbers $k \neq 0$ and t, a *similarity* is a function mapping a location x to $k \times (x - t)$, denoted $[\![k, t]\!]$. Rational number k is called the homothetic factor, and $-k \times t$ is called the translation factor. For simplicity we restrict this definition to the uni-dimensional case; otherwise rotational factors may have to be provided too. Similarities are invertible; they form a group for the law of composition ($[\![k, t]\!]^{-1} = [\![k^{-1}, -k^{-1} \times t]\!]$). Similarities can be extended to positions, by applying the similarity transform to the extracted location.

[2] We will omit G and B most of the time, except in Section 5 where they characterise the number of robots.

```
Definition similarity (k t : ℚ) (p:position) : position := {
  gp := fun n ⇒ k * (p.(gp) n - t) ;
  bp := fun n ⇒ k * (p.(bp) n - t) }.
```

This operation will be (abusively) written $[\![k, t]\!](\texttt{gp} \uplus \texttt{bp})$. Similarities will be used as transformations of frames of reference.

Robograms. We now model what an algorithm r embedded in a correct robot is. For a robot *r-id*$_i$, a computation takes as an input an entire position $\texttt{gp} \uplus \texttt{bp}$ as seen by *r-id*$_i$, in its own frame of reference (scale, origin, etc.),[3] and returns a rational number l_i corresponding to a location (the *destination point*) in the same frame.

Remark 1. Recall that robots in G cannot decide whether another robot is Byzantine, and have no access to a symmetry breaking mechanism such as an identifier. In such a case: the result of r must be invariant by permutations of robots. This is a fundamental property that *any* embedded algorithm must fulfil.

Embedded computation algorithms verifying Remark 1 are called *robograms*, they are naturally defined in our COQ model as follows, two sets (i.e. objects of type `finite`). Note that this definition is completely abstract and makes no use of concrete code whatsoever.

```
Record robogram := {
 algo : position → location ;
 AlgoMorph : ∀ p q σ, (q ≡ p ∘ σ⁻¹) → algo p = algo q }.
```

Computation. So as to provide to r the locations of robots in terms of the considered robot's local frame of reference, and to obtain an absolute location in the *global* coordinate system from the result of r (thus local) we use the notion of similarity. Let us consider a robot *r-id*$_i$ the location of which is at t, and the scale of which is k times the global one, defining a similarity $[\![k, t]\!]$. To obtain the resulting location in terms of the global coordinate system:

1. We center the origin of the position in t, and we zoom according to the homothetic factor k to express the position in the local frame of *r-id*$_i$.
2. The algorithm r computes a local destination point.
3. We apply the inverse of the similarity to obtain the global destination point, that is: according to the global coordinate system.

[3] Note that the scale factor is taken anew at each cycle for *oblivious* robots; in the context of Byzantine failures, it is convenient to consider it as chosen by some adversary.

We denote this operation $r_{[\![k,t]\!]}(\mathtt{gp} \uplus \mathtt{bp}) = [\![k,t]\!]^{-1}(r([\![k,t]\!](\mathtt{gp} \uplus \mathtt{bp})))$. This way we ensure that the global destination point does not depend on the individual frame of reference of robots.[4]

Demons and Properties. A demon provides the position for Byzantine robots, and selects the correct robots to be activated at the current round. As noticed in Footnote 3, we may consider that the demon, acting as an adversary, selects also the scale of the frame of reference for each activated correct robot at each round. A demonic action is thus a record

```
Record demonic_action:= {locate_byz: B → location; frame: G → ℚ}.
```

consisting of a position for Byzantine robots (`locate_byz`), and a function associating to each correct robot a rational number k such that $k = 0$ and the robot is not activated, or $k \neq 0$ and the robot is activated with a scale factor.The actual *demon* is simply an infinite sequence (stream) of demonic actions.

```
CoInductive demon :=  NextDemon: demonic_action → demon → demon.
```

Characteristic properties of demons include *fairness* and synchronous aspects. A demon (seen as a sequence) is locally fair for a robot (inductive property `LocallyFairForOne`) if either this robot is activated during the first demonic action, or if the robot is not activated during the first round but the sequel of the demon is locally fair for that robot. This is related to the classical notion of accessibility. The demon will be fair if it is locally fair for all robots and if its *infinite* sequel is fair.

```
Inductive LocallyFairForOne g (d : demon) : Prop :=
  | ImmediatelyFair : ((demon_head d).frame g) ≠ 0
    → LocallyFairForOne g d
  | LaterFair : ((demon_head d).frame g) = 0
    → LocallyFairForOne g (demon_tail d)
    → LocallyFairForOne g d.

CoInductive Fair (d : demon) : Prop :=
  AlwaysFair : Fair (demon_tail d)
               → (∀ g, LocallyFairForOne g d)
               → Fair d.
```

To be fully synchronous for a demon can be defined similarly. Recall that a fully synchronous demon is a particular case of fair demon such that all correct robots are activated at each round. This is done easily in our setting where we only have to state that the demonic action's `frame` never returns 0. An inductive property `FullySynchronousForOne` states that the first demonic action activates

[4] Note that in this presentation, any considered robot perceives itself as the origin of its local frame of reference

a given robot. A demon is then fully synchronous if `FullySynchronousForOne` holds for all robots and this demon, and if its *infinite* sequel is fully synchronous.

```
CoInductive FullySynchronous d :=
  NextfullySynch: FullySynchronous (demon_tail d)
    → (∀ g, FullySynchronousForOne g d) → FullySynchronous d.
```

Execution. Finally, given an initial position for correct robots $\mathtt{gp}_0$, and a demon

$$D = (\mathtt{locate_byz}_i, \mathtt{frame}_i)_{i\in\mathbb{N}}$$

, we may define an infinite sequence $(\mathtt{gp}_i)_{i\in\mathbb{N}}$ called the *execution* (from $\mathtt{gp}_0$ according to D) as

$$\mathtt{gp}_{i+1}(x) = \begin{cases} r_{[\![\mathtt{frame}_i(x), gp_i(x)]\!]}(\mathtt{gp}_i \uplus \mathtt{bp}_i) & \text{if } \mathtt{frame}_i(x) \neq 0 \\ \mathtt{gp}_i(x) & \text{otherwise} \end{cases}$$

Its type is thus:

```
CoInductive execution :=
  NextExecution : (G → location) → execution → execution.
```

and its computation is reflected by the following corecursive function `execute`:

```
Definition round
  (r : robogram) (da : demonic_action) (gp: G → location) :
    G → location :=
  fun g ⇒
   let k := da.(frame) g in let t := g.(gp) in
   if k = 0 then t
   else t + 1/k * (algo r ([|k,t|]{gp := gp; bp := locate_byz da})).

Definition execute (r : robogram):
  demon → (G → location) → execution :=
    cofix execute d gp :=
      NextExecution gp (execute (demon_tail d) (round r (demon_head d) gp)).
```

5 Case Study: Impossibility Proofs with Byzantine Behaviours

Let us illustrate how well-suited our formalisation is to prove impossibility results, with two theorems by Bouzid *et al.* [5]. Those results address the problem known as *convergence*. Given any initial configuration of robots, the convergence problem requires *correct* robots to approach asymptotically the same, but unknown beforehand, location. That is, for every initial configuration, convergence requires the existence a point c in space such that for every $\varepsilon > 0$, there exists a time τ_ε such that $\forall \tau > \tau_\varepsilon$, all correct robots are within a distance of at most ε of c at τ. The impossibility results in [5] are as follows:

Theorem 1 ([5], Thm 4.3). *It is impossible to achieve convergence if $n \leq 2f$ in the FSYNC uni-dimensional model, where n denotes the number of robots and f denotes the number of Byzantine robots.*

Theorem 2 ([5], Thm 4.4). *Byzantine-resilient convergence is impossible for $n \leq 3f$ in the SSYNC uni-dimensional model and a 2-bounded demon.*

Proofs of Impossibility. Providing a solution to a problem in robot networks usually implies giving a robogram such that the expected property holds at some point in the execution, whatever the demon (seen as an adversary, thus including the Byzantine robots) might do. More precisely, it amounts to showing that there exists a robogram such that for all demons, the property is eventually satisfied. An immediate way of proving such a fact is to provide the actual code for the robogram.

When it comes to impossibility proofs, one has to show instead that for all robogram pretending to be a solution, there exists a demon such that the considered robogram will fail. In fact, the usual attempts to achieve this involve looking for a stronger result: exhibiting a demon that will make any candidate robogram for solution to fail. In both cases the statement of such a result is quantified universally on robograms. Giving any concrete code will not help. However, working with higher-order mechanical theorem proving allows to consider programs as abstract objects and to quantify over them. Robograms will be just characterised by some invariants and the fact that they are supposed to be a solution of a considered problem.

The Theorems in our Formal Model. First of all we need to define formally the convergence problem. In the atomic FSYNC and SSYNC models, an execution $(\mathtt{gp}_i)_{i\in\mathbb{N}}$ is said to be convergent when for any $\varepsilon > 0$ there exists a number of rounds $N_\varepsilon \in \mathbb{N}$ and a location l_ε (in the particular context of [5], $l_\varepsilon \in \mathbb{Q}$) such that for all $n > N_\varepsilon$, all correct robots at round n are no further than ε from l_ε.

$$\forall \varepsilon > 0, \exists N_\varepsilon \in \mathbb{N}, l \in \mathbb{Q}, \forall n > N_\varepsilon, \forall x \in G, |\mathtt{gp}_n(x) - l_\varepsilon| < \varepsilon$$

Convergence expresses that all correct robots will eventually be gathered forever in a disc of radius ε. That is: robots stay gathered *forever* in a disc of radius ε (the coinductive part)...

```
CoInductive imprisonned (prison_center : location) (radius : ℚ)
                        (e : execution) : Prop :=
  InDisk : (∀ g, [(prison_center - execution_head e g)] <= radius)
           → imprisonned prison_center radius (execution_tail e)
           → imprisonned prison_center radius e.
```

. . . disc that they reach eventually (the inductive part)

```
Inductive attracted (pc: location) (radius: ℚ) (e: execution): Prop :=
  | Captured : imprisonned pc radius e → attracted pc radius e
  | WillBeCaptured : attracted pc radius (execution_tail e)
                     → attracted pc radius e.
```

A *solution* to the Convergence problem is a robogram such that for any initial position and assuming a fair demon, the execution eventually imprisons all correct robots.

```
Definition solution (r: robogram) : Prop :=
  ∀ (gp: G → location), ∀ d: demon, Fair d
  → ∀ ε: ℚ, 0 < ε → ∃ lim: location, attracted lim ε (execute r d gp).
```

Remark 2. Our current model considers locations in $\mathbb{Q}$, however the final destination (limit) for convergence is allowed to be in $\mathbb{R} \setminus \mathbb{Q}$, in which case the sequence of l_{ε_i} is a sequence in $\mathbb{Q}$ which has a limit in $\mathbb{R}$.

A formal version of Theorem 1. Let us focus on Theorem 1. As the premises require the demon to be fully-synchronous (FSYNC model) we may as well define what a fully-synchronous demon is, as mentioned on page 10, and specialise with it a version of `solution`. It is worth noticing that our development contains a proof that a fully-synchronous demon is fair and that therefore a solution for any fair scheduler is also a solution for a FSYNC one.

```
Definition solution_FSYNC (r : robogram) : Prop :=
  ∀ (gp : G → location), ∀ (d : demon), FullySynchronous d
  → ∀ ε: ℚ, 0 < ε → ∃ lim: location, attracted lim ε (execute r d gp).
Lemma solution_FAIR_FSYNC : ∀ r, solution r → solution_FSYNC r.
Theorem th1:
  ∀ (g b:finite) (g ≠ ∅) → (r: robogram ({·} ⊎ g) (b ⊎ (g ⊎ {·}))),
  ¬ solution_FSYNC r.
```

It may seem surprising that we use `g` both for correct and Byzantine robots. As a matter of fact, since unions are disjoint by construction, this notation just ensures that the sets of names share the same cardinal. Adding another arbitrary set `b` to the Byzantine part is thus a way of saying that there are at least as many Byzantine robots as correct ones.

Further note that this expression of the theorem clearly states that *there are at least 2 correct robots*; this is not implicit (as no assumption can be in Coq): the considered set of correct robots is indeed a singleton added to a non-empty set.

This theorem and its complete formal proof can be found in our development, as Theorem `no_solution` in File `NoSolutionFSYNC_2f.v`. The file itself is a hundred lines long and relies on various lemmas provided by our framework.

A formal SSYNC fair version of Theorem 2. Akin to the previous theorem the addition of an arbitrary set `b` denotes that the total number of robots is not more than three times the number of Byzantine ones.

We prove in fact a sligthly different result, instead of assuming the demon 2-bounded (that is, the demon may execute a particular robot at most two times between any two executions of *any* other robot [15]), we show that the impossibility result holds for a demon that is fair in SSYNC, and for a number f of Byzantine robots such that $2f < n \leq 3f$ where n is the total number of robots. The bound about f and n by Bouzid *et al.* can be obtained by combining this theorem with the previous one and using lemma `solution_FAIR_FSYNC` above.

```
Theorem th2':
  ∀ (g b: finite) (g ≠ ∅) → (r : robogram ((b ⊎ g) ⊎ g ) (b ⊎ g)),
  ¬ solution r.
```

As before, the theorem and its complete formal proof can be found in our development, as Theorem `no_solution` in File `NoSolutionFAIR_3f.v`. The file itself is 125 lines long and relies on various lemmas provided by our framework.

6 Remarks and Perspectives

The choice of the usual topology of $\mathbb{Q}$ as the basic one is driven by three main reasons. First, it allows arbitrary homotheties (which is not the case for $\mathbb{N}$). Then, it preserves arbitrary precision (thus excluding IEEE754 floating point numbers). Finally, it is axiom-free, while $\mathbb{R}$ is not. As noticed in Remark 2, considering rational numbers is not a handicap for convergence properties.

The total size of our development, including the framework and the proofs of the aforementioned theorems is quite small, as it is approximately 450 lines of specifications and 950 lines of proofs. This is encouraging with reference to how adequate our framework is, as it indicates that proofs are not too intricate and remain human readable.

It is worth noticing that our formalism is robust enough to take into account several alternative models with few modifications. For instance, and thanks to the high abstraction level of our framework, considering a multi-dimensional space (instead of just a line) only amounts to considering tuples for locations (and not simply rational numbers) and adding a rotation for some similarities. The effort is thus put on the actual proof and not on the modeling tasks. Hence, a first short-term perspective is to tackle impossibility proofs for convergence on the rational plane or three dimensional space. Similarly, going from strong multiplicity to weak multiplicity is only a redefinition of the equality relation between positions... The same remark applies to demons' characteristics. Adding constraints such as being fully-synchronous is just *(i)* Defining this constraint,

and *(ii)* Adding this constraint as an assumption in the statement of a theorem. Of course proofs may be very demanding in all those models, but we want to emphasise that relevant adaptations of our framework are rather non-expensive.

An noteworthy added benefit of our abstract formalisations is that keeping them as general as possible may lead to relaxing premises of theorems, thus potentially discovering new results (*e.g.* formalizing weaker daemons [15] and weaker forms of Byzantine behaviours could lead to stronger impossibility results).

Finally, we plan to use our development for positive results also, that is, to prove properties of concrete algorithms. The language of Coq can handle datatypes, programs, and properties about them. Our general framework should allow for certification of embedded algorithms, as both concrete code for robots and global properties of the network fit in. Notice that such proofs would guarantee the expected properties in infinite spaces, *i.e.* without limits on locations.

References

1. Noa Agmon and David Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM Journal on Computing*, 36(1):56–82, 2006.
2. José Bacelar Almeida, Manuel Barbosa, Endre Bangerter, Gilles Barthe, Stephan Krenn, and Santiago Zanella Béguelin. Full proof cryptography: verifiable compilation of efficient zero-knowledge protocols. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM Conference on Computer and Communications Security*, pages 488–500. ACM, 2012.
3. Marc Bezem, Roland Bol, and Jan Frisco Groote. Formalizing process algebraic verifications in the calculus of constructions. *Formal Aspects of Computing*, 9:1–48, 1997.
4. François Bonnet, Xavier Défago, Franck Petit, Maria Potop-Butucaru, and Sébastien Tixeuil. Brief announcement: Discovering and assessing fine-grained metrics in robot networks protocols. In *Proceedings of the International Conference on Stabilization, Safety, and Security in Distributed Systems (SSS 2012)*, Lecture Notes in Computer Science, Toronto, Canada, October 2012. Springer-Verlag.
5. Zohir Bouzid, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. Optimal byzantine-resilient convergence in uni-dimensional robot networks. *Theoretical Computer Science*, 411(34-36):3154–3168, 2010.
6. Michaël Cadilhac, Thomas Hérault, Richard Lassaigne, Sylvain Peyronnet, and Sébastien Tixeuil. Evaluating complex MAC protocols for sensor networks with APMC. *Electronic Notes in Theoretical Computer Science*, 185:33–46, July 2007.
7. Dominique Cansell and Dominique Méry. *Logics of Specification Languages*, chapter The Event-B Modelling Method: Concepts and Case Studies, pages 47–152. Springer-Verlag, 2007.
8. Pierre Castéran, Vincent Filou, and Mohamed Mosbah. Certifying distributed algorithms by embedding local computation systems in the coq proof assistant. In Adel Bouhoula and Tetsuo Ida, editors, *Symbolic Computation in Software Science (SCSS'09)*, 2009.
9. Ching-Tsun Chou. Mechanical verification of distributed algorithms in higher-order logic. *The Computer Journal*, 38:158–176, 1995.
10. Julien Clément, Carole Delporte-Gallet, Hugues Fauconnier, and Mihaela Sighireanu. Guidelines for the verification of population protocols. In *ICDCS*, pages 215–224, Minneapolis, Minnesota, USA, June 2011. IEEE Computer Society.
11. Thierry Coquand and Christine Paulin-Mohring. Inductively defined types. In Per Martin-Löf and Grigori Mints, editors, *Proceedings of Colog'88*, volume 417 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.
12. Denis Cousineau, Damien Doligez, Leslie Lamport, Stephan Merz, Daniel Ricketts, and Hernán Vanzetto. TLA + Proofs. In Dimitra Giannakopoulou and Dominique Méry, editors, *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 147–154, Paris, France, August 2012. Springer-Verlag.
13. Yuxin Deng and Jean-François Monin. Verifying self-stabilizing population protocols with coq. In Wei-Ngan Chin and Shengchao Qin, editors, *Third IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE 2009)*, pages 201–208, Tianjin, China, July 2009. IEEE Computer Society.
14. Stéphane Devismes, Anissa Lamani, Franck Petit, Pascal Raymond, and Sébastien Tixeuil. Optimal grid exploration by asynchronous oblivious robots. In *Proceedings of the International Conference on Stabilization, Safety, and Security in Distributed Systems (SSS 2012)*, Lecture Notes in Computer Science, Toronto, Canada, October 2012. Springer-Verlag.
15. Swan Dubois and Sébastien Tixeuil. A taxonomy of daemons in self-stabilization. Technical Report 1110.0334, ArXiv eprint, October 2011.

16. Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.
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. Wan Fokkink. *Modelling Distributed Systems*. EATCS Texts in Theoretical Computer Science. Springer-Verlag, 2007.
19. Georges Gonthier. Formal proof—the four- color theorem. In *Notices of the AMS*, volume 55, page 1370. decmber 2008.
20. Georges Gonthier. Engineering mathematics: the odd order theorem proof. In Roberto Giacobazzi and Radhia Cousot, editors, *POPL*, pages 1–2. ACM, 2013.
21. Philipp Küfner, Uwe Nestmann, and Christina Rickmann. Formal verification of distributed algorithms - from pseudo code to checked proofs. In Jos C. M. Baeten, Thomas Ball, and Frank S. de Boer, editors, *IFIP TCS*, volume 7604 of *Lecture Notes in Computer Science*, pages 209–224, Amsterdam, The Netherlands, September 2012. Springer-Verlag.
22. Leslie Lamport. Byzantizing paxos by refinement. In David Peleg, editor, *DISC*, volume 6950 of *Lecture Notes in Computer Science*, pages 211–224. Springer, 2011.
23. Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.
24. Xavier Leroy. A formally verified compiler back-end. *Journal of Automated Reasoning*, 43(4):363–446, 2009.
25. Igor Litovsky, Yves Métivier, and Éric Sopena. Graph relabelling systems and distributed algorithms. In Hartmut Ehrig, Hans-Jörg Kreowski, Ugo Montanari, and Grzegorz Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 3, pages 1–56. World Scientific, 1999.
26. John Mccarthy and James Painter. Correctness of a compiler for arithmetic expressions. In *Proceedings of Applied Mathematica*, volume 19 of *Mathematical Aspects of Computer Science*, pages 33–41. American Mathematical Society, 1967.
27. Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.
28. Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal of Computing*, 28(4):1347–1363, 1999.
29. Laurent Théry and Guillaume Hanrot. Primality proving with elliptic curves. In Klaus Schneider and Jens Brandt, editors, *20th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2007)*, volume 4732 of *Lecture Notes in Computer Science*, pages 319–333, Kaiserslautern, Germany, September 2007. Springer-Verlag.