# Certification of Distributed Self-Stabilizing Algorithms Using Coq
*Master Internship*

Karine Altisen Pierre Corbineau Stéphane Devismes

**Verimag Lab**
**Contacts:** {Karine.Altisen, Pierre.Corbineau, Stephane.Devismes}@imag.fr

**Scientific Context.** Modern distributed systems can be *large-scale* (*e.g.*, Internet), *dynamic* (*e.g.*, Peer-to-Peer systems), and / or *resource constrained* (*e.g.*, WSNs — Wireless Sensor Networks). Those characteristics increase the number of faults which may hit the system. For instance, in WSNs, processes are subject to crash failures because of their limited battery. Moreover, their communications use radio channels which are subject to intermittent loss of messages. Now, due to their large-scale and the adversarial environment where they may be deployed, intervention to repair them cannot be always envisioned. In this context, fault-tolerance, *i.e.*, the ability of a distributed algorithm to endure the faults by itself, is mandatory.

Self-stabilization is a versatile lightweight technique to withstand transient faults in a distributed system. After transient faults hit and place the system into some arbitrary global state, a self-stabilizing algorithm returns, in finite time, to a correct behavior without external intervention. Self-stabilization makes no hypotheses on the nature or extent of transient faults that could hit the system, and recovers from the effects of those faults in a unified manner.

Today's researches on self-stabilizing algorithms focus on more and more complex problems and adversarial environments. This makes the proof that an algorithm actually achieves self-stabilization even more complex and subtle to establish. Now, those proofs are usually performed by hand, using informal reasoning. Such methods are clearly pushed to their limits and this justifies the use of a *proof assistant*.

Proof assistants are environments in which a user can express programs, state theorems, and develop proofs interactively, those ones being mechanically checked (*i.e.*, machine-checked). In particular, the Coq proof assistant [4], which is targeted by this project, has been successfully used for various tasks such as mathematical developments as involved as the 4-colors or Feit-Thompson theorems, formalization of programming language semantics leading to the certification of a C compiler, certified numerical libraries, and verification of cryptographic protocols.

However, despite the clear benefits of formalized proofs in the area of self-stabilization, this topic remains almost unexplored until now. Courtieu [1] provides a general setting for reasoning on self-stabilization in Coq. A formal correctness proof of Dijkstra's seminal self-stabilizing algorithm has already been conducted with the PVS proof assistant [3]. This experiment ended with an elegant proof but the case of Dijkstra's two rules algorithm for token circulation on a ring is too simple to draw a general conclusion.

**Subject.** The long term goal of this project is to propose a *framework, based on Coq, to (semi-) automatically construct certified proofs of self-stabilization*. To do so, we are guided by case studies, in particular the one based on a non-trivial algorithm [2]. This work requires to import into Coq the computational model in which the targeted algorithm is designed, to formalize the algorithm itself and its specification. Then the algorithm has to be proved in Coq: safety and convergence from the specification and also some performance results of the algorithm.

Algorithms implementing such properties often share typical features. For example, the design of those algorithms is usually compositional: two simple algorithms can be used together (composed) to create a

more complex one. For example also, proof schemes such as induction principle and well-founded order are usually the basis for convergence proofs. We henceforth aim at developing Coq libraries of results about *e.g.*, composition of algorithms or convergence results and then to use them under case studies.

An existing Coq library is under development that involves all of these facets; the activity proposed for this internship is to contribute to the library as detailed above. Precisely, the subject requires:

- A bibliographical study: deep understanding of the algorithm used as case-study [2].
- To become familiar with the existing Coq library, learning some Coq skills, if necessary.
- To develop further the existing Coq library under one of the following directions: composition of algorithms and tools to prove their safety and convergence; tools for proving performance results of algorithms; development and application to the case study.

**Required Skills.** An important background about sequential algorithmic, in particular proof of algorithms, is mandatory. This subject also requires background about distributed systems and formal methods.

**Working context.** The internship is part of AGIR-UGA project PADEC. The student will be integrated in the lab Verimag[1].

Possible extensions into a PhD thesis.

*2015-2016*

# References


[1] P. Courtieu. Proving Self-Stabilization with a Proof Assistant. In *IPDPS*. IEEE Computer Society, 2002.

[2] Ajoy Kumar Datta, Lawrence L. Larmore, Stéphane Devismes, Karel Heurtefeux, and Yvan Rivierre. Competitive self-stabilizing k-clustering. In *ICDCS, 2012 IEEE 32nd International Conference on Distributed Computing Systems, Macau, China, June 18-21, 2012*, pages 476–485. IEEE, 2012.

[3] Sandeep S. Kulkarni, John M. Rushby, and Natarajan Shankar. A case-study in component-based mechanical verification of fault-tolerant programs. In *WSS*, pages 33–40, 1999.

[4] The Coq Development Team. *The Coq Proof Assistant Documentation*, June 2012.


[1] http://www-verimag.imag.fr/