# Brief Announcement: Model Checking Rendezvous Algorithms for Robots with Lights in Euclidean Space

Xavier Défago, Adam Heriban, Sébastien Tixeuil, Koichi Wada

# Brief Announcement: Model Checking Rendezvous Algorithms for Robots with Lights in Euclidean Space

**Xavier Défago** ⬛
School of Computing, Tokyo Institute of Technology, Japan

**Adam Heriban**
Sorbonne Université, CNRS, LIP6, Paris, France

**Sébastien Tixeuil** ⬛
Sorbonne Université, CNRS, LIP6, Paris, France

**Koichi Wada** ⬛
Faculty of Science and Engineering, Hosei University, Tokyo, Japan

──── **Abstract** ────

This announces the first successful attempt at using model-checking techniques to verify the correctness of self-stabilizing distributed algorithms for robots evolving in a *continuous* environment. The study focuses on the problem of rendezvous of two robots with lights and presents a generic verification model for the SPIN model checker. It will be presented in full at an upcoming venue.

## Introduction

Following the seminal work of Suzuki and Yamashita [9], we model robots as points in the 2D Euclidean plane that independently execute their own instance of the same deterministic algorithm. Robots are anonymous, oblivious, and disoriented (i.e., no common coordinate system), and repeatedly execute Look-Compute-Move (LCM) cycles, where a robot "Looks" at its surroundings, obtains a snapshot of the locations of all robots, "Computes" a destination, and "Moves" toward it. Additionally, robots are equipped with a light that can emit a color among a fixed number of distinct colors [1]. A robot observes all lights during its Look phase and changes its light at the end of its Compute phase.

The literature [9, 3] considers three main levels of synchrony: FSYNC model [9] where every LCM cycle is performed simultaneously by all robots; SSYNC where each cycle may be skipped by a subset of the robots; and ASYNC which imposes no restriction on synchronization between the robots, thus allowing robots to be observed while moving.

The problem of gathering requires all robots to reach a single point in finite time, regardless of their initial locations. The particular case of gathering two robots is called *rendezvous*. In SSYNC, no deterministic algorithm can solve rendezvous without additional assumptions [9]. This case being trickier than three or more robots due to the inherent symmetry of observed configurations. A rendezvous algorithm is self-stabilizing if robots eventually reach and

stay forever at the same location regardless of the *initial configuration.* Algorithms that set constraints on the initial configuration (*e.g.*, must start with a specific color) are not self-stabilizing.

Viglietta [10] proved that two colors are sufficient in SSYNC and that three colors are necessary and sufficient in ASYNC for a specific class of algorithms. Okumura et al. [8] presented rendezvous algorithms when additional restrictions are made on the model (rigid moves, initial colors, etc.). Finally, Heriban et al. [5] showed that two colors are necessary and sufficient in ASYNC without extra assumptions.

As they use case-based reasoning, the handwritten proofs of rendezvous algorithms with lights are lengthy, complex, tedious to check and write, and hence highly error-prone. The aim of the work is hence to automate the process and verify the correctness of these self-stabilizing rendezvous algorithms using model checking.

Model checking relies on the definition of a verification model consisting exclusively of finite values. In particular, the domain of variables must be kept to a minimum. In the case of rendezvous of robots with lights, the number of robots and the number of colors are finite and very small. However, the position of the robots and the time of their activations have all infinite domains. This means that the verification model must capture the important characteristics of the original model while representing everything in a tractable state space. In other words, the only way to obtain automated proofs of correctness in the continuous space context through model checking is to use a more abstract verification model.

### Methodology

The approach consists of two parts: (a) the definition of the verification model and its correctness, and (b) the implementation of the verification model and algorithms in the model-checker.

First, we prove several important results that are used to support the verification model. In particular, there is obviously an inherent loss of generality when representing infinite domains with finite values. In this case, our verification model is designed to fail conservatively, in the sense that an incorrect algorithm must always result in a failed verification, whereas a correct algorithm may not always result in a successful one. This point is crucial because this is why this approach is sound.

The key aspect of the verification model consists in representing the Euclidean environment with only two discrete values (or three depending on assumptions) with an appropriate way to resolve movements. The difficult and subtle part is obviously the latter one and is presented in details in the full version of this manuscript [2].

Second, we express the verification model in the SPIN model checker [6]. Given a model expressed in the Promela language (a concurrent language reminiscent of Hoare's CSP) and a linear temporal logic formula, the model checker explores every possible paths of the model until it reaches an invalid state (one that violates the formula) from an initial state, in which case it reports that path as a counter-example execution that violates the temporal logic formula. Otherwise, it reports success. Since the model checker performs an *exhaustive search*, branching out at every non-deterministic choice, the condition is then proved to hold. Model checking has gone a long way since the 70s when it began, and current solvers are now able to handle millions of states without much problem.

We wrote the Promela model to be as modular as possible. In particular, the same framework is used to model seven different synchrony models, including FSYNC, SSYNC, and ASYNC. The rendezvous algorithms are expressed as functions mapping an observation to a color and a move. Based on a theorem proved in the first part, a fair execution can be modeled by limiting the number of consecutive activations by a constant value.

■ **Table 1** Results of model-checking liveness of some known algorithms.

| Central. | FSYNC | SSYNC | LC-atom. ASYNC | Move-atom. ASYNC | ASYNC | |
|:---:|:---:|:---:|:---:|:---:|:---:|:---|
| - | - | - | - | - | - | Neither robots move |
| - | ✓ | - | - | - | - | Move to midpoint |
| ✓ | - | - | - | - | - | Move to other |
| ✓ | ✓ | ✓ | ✓ | - | - | Viglietta [10] 2 colors |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | Viglietta [10] 3 colors |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | Heriban+ [5] 2 colors |
| ✓ | ✓ | ✓ | - | - | - | Flocchini+ [4] 3 colors |
| ✓ | ✓ | ✓ | ✓ | - | - | Okumura+ [7] 5 colors |
| ✓ | - | - | - | - | - | Okumura+ [7] 4 col.; quasi-SS |
| ✓ | - | - | - | - | - | Okumura+ [7] 3 col.; non-SS |

## Validation

To assess the verification model and its SPIN implementation, we have checked ten different rendezvous algorithms: three trivial baseline algorithms as well as seven known algorithms from the literature. For each algorithms, two of which are not self-stabilizing, it is widely-known in which models they achieve rendezvous or fail. Table 1 summarizes the results (either positive or negative with a counter-example) obtained from model-checking these algorithms in six different synchrony models, including the three most common (FSYNC, SSYNC, and ASYNC). Each outcome of the resulting 60 tests is consistent with the literature, and the entire test runs in a few minutes on a regular laptop.

## References

**1** S. Das, P. Flocchini, G. Prencipe, N. Santoro, and M. Yamashita. Autonomous mobile robots with lights. *Theor. Comput. Sci.*, 609:171–184, 2016. `doi:10.1016/j.tcs.2015.09.018`.

**2** X. Défago, A. Heriban, S. Tixeuil, and K. Wada. Using Model Checking to Formally Verify Rendezvous Algorithms for Robots with Lights in Euclidean Space. CoRR abs/1907.09871, arXiv, July 2019. `arXiv:1907.09871`.

**3** P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of asynchronous robots with limited visibility. *Theor. Comput. Sci.*, 337(1-3):147–168, 2005. `doi:10.1016/j.tcs.2005.01.001`.

**4** P. Flocchini, N. Santoro, G. Viglietta, and M. Yamashita. Rendezvous with constant memory. *Theor. Comput. Sci.*, 621:57–72, 2016. `doi:10.1016/j.tcs.2016.01.025`.

**5** A. Heriban, X. Défago, and S. Tixeuil. Optimally Gathering Two Robots. In *Proc. ICDCN*, pages 3:1–3:10, January 2018. `doi:10.1145/3154273.3154323`.

**6** G. Holzmann. *The SPIN Model Checker: Primer & Reference Manual.* Addison-Wesley, 2004.

**7** T. Okumura, K. Wada, and X. Défago. Optimal Rendezvous $\mathcal{L}$-Algorithms for Asynchronous Mobile Robots with External-Lights. In *Proc. OPODIS*, pages 24:1–16, December 2018. `doi:10.4230/LIPIcs.OPODIS.2018.24`.

**8** T. Okumura, K. Wada, and Y. Katayama. Brief Announcement: Optimal Asynchronous Rendezvous for Mobile Robots with Lights. In *Proc. SSS*, pages 484–488, November 2017. `doi:10.1007/978-3-319-69084-1_36`.

**9** I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *SIAM J. Comput.*, 28(4):1347–1363, 1999. `doi:10.1137/S009753979628292X`.

**10** G. Viglietta. Rendezvous of Two Robots with Visible Bits. In *Proc. ALGOSENSORS*, pages 291–306, September 2013. `doi:10.1007/978-3-642-45346-5_21`.