PEREGRINE: A Tool for the Analysis of Population Protocols *

Michael Blondin[0000−0003−2914−2734], Javier Esparza[0000−0001−9862−4919], and Stefan Jaax[0000−0001−5789−8091]

Technische Universität München
{blondimi, esparza, jaax}@in.tum.de

Abstract. We introduce PEREGRINE, the first tool for the analysis and parameterized verification of population protocols. Population protocols are a model of computation very much studied by the distributed computing community, in which mobile anonymous agents interact stochastically to achieve a common task. PEREGRINE allows users to design protocols, to simulate them both manually and automatically, to gather statistics of properties such as convergence speed, and to verify correctness automatically. This paper describes the features of PEREGRINE and their implementation.

Keywords: population protocols, distributed computing, parameterized verification, simulation.

1 Introduction

Population protocols [4, 1, 3] are a model of distributed computing in which replicated, mobile agents with limited computational power interact stochastically to achieve a common task. They provide a simple and elegant formalism to model, e.g., networks of passively mobile sensors [1, 5], trust propagation [13], evolutionary dynamics [14], and chemical systems, under the name chemical reaction networks [16, 19, 12].

Population protocols are parameterized: the number of agents does not change during the execution of the protocol, but is *a priori* unbounded. A protocol is correct if it behaves correctly for all of its infinitely many initial configurations. For this reason, it is challenging to design correct and efficient protocols.

In this paper we introduce PEREGRINE[1], the first tool for the parameterized analysis of population protocols. PEREGRINE is intended for use by researchers in distributed computing and systems biology. It allows the user to specify protocols either through an editor or as simple scripts, and to analyze them via a

* M. Blondin was supported by the Fonds de recherche du Québec – Nature et technologies (FRQNT).

[1] PEREGRINE can be found at https://peregrine.model.in.tum.de.

graphical interface. The analysis features of PEREGRINE include manual step-by-step simulation; automatic sampling; statistics generation of average convergence speed; detection of incorrect executions through simulation; and formal verification of correctness. The first four features are supported for all protocols, while verification is supported for silent protocols, a large subclass of protocols [6]. Verification is performed automatically over *all* of the infinitely many initial configurations using the recent approach of [6] for solving the so-called well-specification problem.

Related work. The problem of automatically verifying that a population protocol conforms to its specification for *one fixed initial configuration* has been considered in [10, 11, 17, 20]. In [10], *ad hoc* search algorithms are used. In [11, 17], the authors show how to model the problem in the probabilistic model checker PRISM, and under certain conditions in SPIN. In [20], the problem is modeled with the PAT toolkit for model checking under fairness assumptions. All these tools increase our confidence in the correctness of a protocol. However, compared to PEREGRINE, they are not visual tools, they do not offer simulation capabilities, and they can only verify the correctness of a protocol for a finite number of initial configurations, with typically a small number of agents. PEREGRINE proves correctness for all of the infinitely many initial configurations, with an arbitrarily large number of agents.

As mentioned in the introduction, population protocols are isomorphic to chemical reaction networks (CRNs), a popular model in natural computing. Cardelli et al. have recently developed model checking techniques and analysis algorithms for *stochastic* CRNs [8, 7, 9]. The problems studied therein are incomparable to the parameterized questions addressed by PEREGRINE.

The verification algorithm of PEREGRINE is based on [6], where a novel approach for the parameterized verification of silent population protocols has been presented. The command-line tool of [6] only offers support for proving correctness, with no functionality for visualization or simulation. Further, contrary to PEREGRINE, the tool cannot produce counterexamples when correctness fails.

2 Population protocols

We introduce population protocols through a simple example and then briefly formalize the model. We refer the reader to [4] for a more thorough but still intuitive presentation. Suppose anonymous and mobile agents wish to take a majority vote. Intuitively, *anonymous* means that agents have no identity, and *mobile* that agents are "wandering around", and can only interact whenever they bump into each other. In order to vote, all agents conduct the following protocol. Each agent is in one out of four states $\{Y, N, y, n\}$. Initially all agents are in the states Y or N, corresponding to how they want to vote (states y, n are auxiliary states). Agents repeatedly interact pairwise according to the following rules:

$$a\colon YN \mapsto yn \qquad b\colon Yn \mapsto Yy \qquad c\colon Ny \mapsto Nn \qquad d\colon yn \mapsto yy$$

For example, if the population initially has two agents of opinion "yes" and one agent of opinion "no", then a possible execution is:

$$\{\!|\underline{Y}, Y, \underline{N}|\!\} \xrightarrow{a} \{\!|y, \underline{Y}, \underline{n}|\!\} \xrightarrow{b} \{\!|y, Y, y|\!\}, \tag{1}$$

where e.g. $\{\!|Y, Y, N|\!\}$ denotes the multiset with two agents in state Y and one agent in state N.

The goal of every population protocol is to ensure that the agents eventually reach a lasting consensus, i.e., a multiset in which (1) either all agents are in "yes"-states, or all agents are in "no"-states, and (2) further interactions do not destroy the consensus. On top of this universal specification, each protocol has an individual goal, determining which initial configurations should reach the "yes" and the " no" lasting consensus. In the majority protocol above, the agents should reach a "yes"-consensus iff 50% or more agents vote "yes".

Execution (1) above leads to a lasting "yes"-consensus; further, the consensus is the right one, since 2 out of 3 agents voted "yes". In fact, assuming agents interact uniformly and independently at random, the above protocol is correct: executions almost surely reach a correct lasting consensus.

More formally, a population protocol is a tuple (Q, T, I, O) where Q is a finite set of *states*, $T \subseteq Q^2 \times Q^2$ is a set of *transitions*, $I \subseteq Q$ are the *initial states* and $O\colon Q \to \{0, 1\}$ is the *output mapping*. A *configuration* is a non-empty multiset over Q, an *initial configuration* is a non-empty multiset over I, and a configuration is *terminal* if it cannot be altered by any transition. A configuration is in a *consensus* if all of its states map to the same output under O.

An *execution* is a finite or infinite sequence $C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} \cdots$ such that C_i is obtained from applying transition t_i to C_{i-1}. A *fair execution* is either a finite execution that reaches a terminal configuration, or an infinite execution such that if $\{i \in \mathbb{N} : C_i \xrightarrow{*} D\}$ is infinite, then $\{i \in \mathbb{N} : C_i = D\}$ is infinite for any configuration D. In other words, fairness ensures that a configuration cannot be avoided forever if it is reachable infinitely often. Fairness is an abstraction of the random interactions occurring within a population. A configuration C is in a *lasting consensus* if every execution from C only leads to configurations of the same consensus.

If for every initial configuration C, all fair executions from C lead to a lasting consensus $\varphi(C) \in \{0, 1\}$, then we say that the protocol *computes* the predicate φ. For example, the above majority protocol with $O(Y) = O(y) = 1$ and $O(N) = O(n) = 0$ computes the predicate $C[Y] \geq C[N]$, where $C[x]$ denotes the number of occurrences of state x in C. A protocol does not necessarily compute a predicate. For example, if we alter the majority protocol by removing transition d, then $\{\!|Y, N|\!\} \xrightarrow{a} \{\!|y, n|\!\}$ is a fair execution, but $\{\!|y, n|\!\}$ is not in a consensus. In other words, transition d acts as a tie-breaker which allows to reach the consensus configuration $\{\!|y, y|\!\}$. A protocol that computes a predicate is said to be *well-specified*. It is well-known that well-specified population protocols compute precisely the predicates definable in Presburger arithmetic [3]. On top of different *majority protocols* for the predicate $C[x] \geq C[y]$, the literature contains, e.g., different families of so-called *flock-of-birds protocols* for the predicates $C[x] \geq c$,

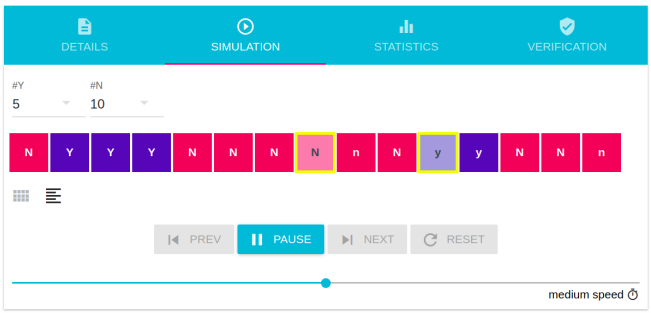

Fig. 1. Simulation of the majority protocol from the initial configuration $\wr 5 \cdot Y, 10 \cdot N \wr$.

where c is an integer constant, and families of *threshold protocols* for the predicates $a_1 \cdot C[x_1] + \cdots + a_n \cdot C[x_n] \geq c$, where $a_1, \ldots, a_n, c$ are integer constants and $x_1, \ldots, x_n$ are initial states.

3 Analyzing population protocols

Peregrine is a web tool with a JavaScript frontend and a Haskell backend. The backend makes use of the SMT solver Z3 [15] to test satisfiability of Presburger arithmetic formulas. The user has access to four main features through the graphical frontend. We present these features in the remainder of the section.

Protocol description. Peregrine offers a description language for both single protocols and families of protocols depending on some parameters. Single protocols are described either through a graphical editor or as simple Python scripts. Families of protocols (called parametric protocols) can only be specified as scripts, but Peregrine assists the user by generating a code skeleton.

Simulation. Population protocols can be simulated through a graphical player depicted in Figure 1. The user can pick an initial configuration and simulate the protocol by either manual selection of interactions, or by letting a scheduler pick interactions uniformly at random. The simulator keeps a history of the execution which can be rewound at any time, making it easy to experiment with the different behaviours of a protocol. Configurations can be displayed in two ways: either as explicit populations, as illustrated in Figure 1, or as bar charts of the states count, more convenient for large populations.

Statistics. Peregrine can generate statistics from batch simulations. The user provides four parameters: $s_{\min}, s_{\max}, m$ and n. Peregrine generates n random executions as follows. For each execution, a number s is picked uniformly at random from $[s_{\min}, s_{\max}]$, and an initial configuration of size s is then picked uniformly at random. Each step of an execution is picked uniformly at random among enabled interactions. If no terminal configuration is reached within m

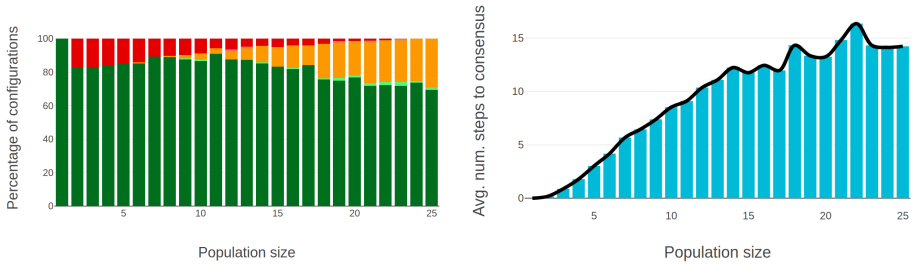

Fig. 2. Statistics for 5000 random executions of the approximate majority protocol of [2], of length at most 40, from initial configurations of size at most 25. The left plot shows the percentage of executions reaching a consensus (dark green: lasting correct, light green: correct, light red: incorrect, dark red: lasting incorrect) and no consensus (orange). In this example the occurrences of light red are negligible. The right plot shows the average number of steps to convergence.

steps, then the simulation halts. In the end, n executions of length at most m are gathered. PEREGRINE classifies the generated executions according to their consensus, and computes statistics on the convergence speed (see the next two paragraphs). The results can be visualized in different ways, and the raw data can be exported as a JSON file.

Consensus. For each random execution, PEREGRINE checks whether the last configuration of an execution is in a consensus and, if so, whether the consensus corresponds to the expected output of the protocol. PEREGRINE reports which percentage of the executions reach a consensus, and whether the consensus is correct and/or lasting. In normal mode, PEREGRINE only classifies an execution as lasting consensus if it ends in a terminal configuration. In the *increased accuracy* mode, if the execution ends in a configuration C of consensus $b \in \{0,1\}$, then the model checker LOLA [18] is used to determine whether there exists a configuration C' such that $C \xrightarrow{*} C'$ and C' is not of consensus b. If it is not the case, then PEREGRINE concludes that C is in a lasting consensus. PEREGRINE plots the percentage of executions in each category as a function of the population size, as illustrated on the left of Figure 2.

Average convergence speed. PEREGRINE also provides statistics on the convergence speed of a protocol. Let $C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} \cdots \xrightarrow{t_\ell} C_\ell$ be an execution such that C_ℓ is in a consensus $b \in \{0,1\}$. The *number of steps to convergence* of the execution is defined as 0 if all configurations are of consensus b, and otherwise as $i+1$, where i is the largest index such that C_i is not in consensus b. For each population size, PEREGRINE computes the average number of steps to convergence of all consensus executions of that population size, and plots the information as illustrated on the right of Figure 2.

Verification. PEREGRINE can automatically verify that a population protocol computes a given predicate. Predicates can be specified by the user in

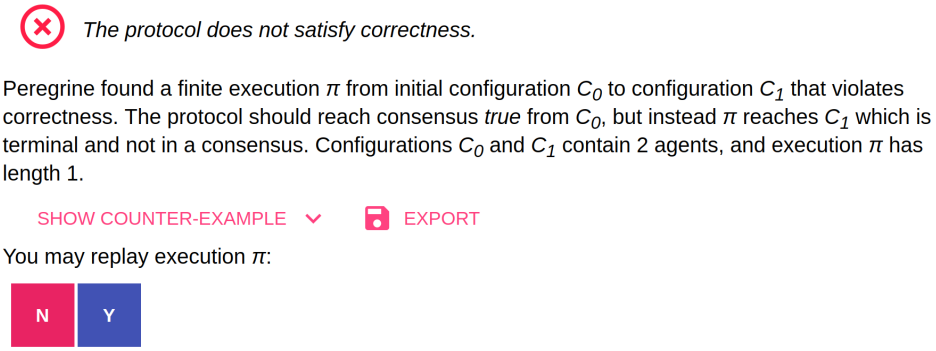

Fig. 3. Verification of the majority protocol of Section 2 without transition $d\colon yn \mapsto yy$.

quantifier-free Presburger arithmetic extended with the family of predicates $\{x \equiv y \pmod{c}\}_{c \geq 2}$, which is equivalent to Presburger arithmetic. For example, for the majority protocol of Section 2, the user simply specifies `C[Y] >= C[N]`.

PEREGRINE implements the approach of [6] to verify correctness of protocols which are silent. A protocol is said to be *silent* if from every initial configuration, every fair execution leads to a terminal configuration. The majority protocol of Section 2 and most existing protocols from the literature are silent [6]. We briefly describe the approach of [6] and how it is integrated into PEREGRINE.

Suppose we are given a population protocol $\mathcal{P}$ and we wish to determine whether it computes a predicate φ. The procedure first tries to prove that $\mathcal{P}$ is silent. This is done by verifying a more restricted condition called *layered termination*. Verifying the latter property reduces to testing satisfiability of a Presburger arithmetic formula. If this formula holds, then the protocol is silent, otherwise no conclusion is derived. However, essentially all existing silent protocols satisfy layered termination [6].

Once $\mathcal{P}$ is proven to be silent, the procedure attempts to prove that no "bad execution" exists. More precisely, it checks whether there exist configurations C_0 and C_1 such that $C_0 \xrightarrow{*} C_1$, C_0 is initial, C_1 is terminal, and C_1 is not in consensus $\varphi(C_0) \in \{0, 1\}$. Since reachability is not definable in Presburger arithmetic, a Presburger-definable over-approximation $\xrightarrow[]{*}$ of reachability, borrowed from Petri net theory, is used instead. We obtain the following formula $\Phi_{\text{bad-exec}}$:

$$\exists C_0, C_1 \colon C_0 \xrightarrow{*} C_1 \wedge \bigwedge_{q \notin I} C_0[q] = 0 \wedge \bigwedge_{t \in T} \operatorname{succ}(C_1, t) \subseteq \{C_1\} \wedge \bigvee_{q \in C_1} (O(q) = \neg\varphi(C_0)).$$

If $\Phi_{\text{bad-exec}}$ is unsatisfiable, then $\mathcal{P}$ is correct. Otherwise, no conclusion is reached, and $\Phi_{\text{bad-exec}}$ is iteratively strengthened by enriching the over-approximation $\xrightarrow{*}$. Whenever $\Phi_{\text{bad-exec}}$ is satisfied by (C_0, C_1), PEREGRINE calls the model-checker LOLA to test whether C_1 is indeed reachable from C_0. If so, then PEREGRINE reports $\mathcal{P}$ to be incorrect, and generates a counter-example execution, which can be replayed or exported as a JSON file (see Figure 3).

Currently PEREGRINE can verify protocols with up to a hundred states and a few thousands transitions. The bottleneck is the size of the constraint system. Due to lack of space, we refer the reader to [6] for detailed experimental results.

References

1. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. In: Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC). pp. 290–299 (2004). https://doi.org/10.1145/1011767.1011810
2. Angluin, D., Aspnes, J., Eisenstat, D.: A simple population protocol for fast robust approximate majority. Distributed Computing **21**(2), 87–102 (2008). https://doi.org/10.1007/s00446-008-0059-z
3. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distributed Computing **20**(4), 279–304 (2007). https://doi.org/10.1007/s00446-007-0040-2
4. Aspnes, J., Ruppert, E.: An introduction to population protocols. In: Middleware for Network Eccentric and Mobile Applications, pp. 97–120. Springer-Verlag (2009). https://doi.org/10.1007/978-3-540-89707-1_5
5. Beauquier, J., Blanchard, P., Burman, J., Delaët, S.: Tight complexity analysis of population protocols with cover times - the ZebraNet example. Theoretical Computer Science **512**, 15–27 (2013). https://doi.org/10.1016/j.tcs.2012.10.032
6. Blondin, M., Esparza, J., Jaax, S., Meyer, P.J.: Towards efficient verification of population protocols. In: Proc. 36th ACM Symposium on Principles of Distributed Computing (PODC). pp. 423–430 (2017). https://doi.org/10.1145/3087801.3087816
7. Cardelli, L., Ceska, M., Fränzle, M., Kwiatkowska, M.Z., Laurenti, L., Paoletti, N., Whitby, M.: Syntax-guided optimal synthesis for chemical reaction networks. In: Proc. 29th International Conference Computer Aided Verification (CAV). pp. 375–395 (2017). https://doi.org/10.1007/978-3-319-63390-9_20
8. Cardelli, L., Kwiatkowska, M., Laurenti, L.: Stochastic analysis of chemical reaction networks using linear noise approximation. Biosystems **149**, 26–33 (2016). https://doi.org/10.1016/j.biosystems.2016.09.004
9. Cardelli, L., Tribastone, M., Tschaikowski, M., Vandin, A.: Syntactic Markovian bisimulation for chemical reaction networks. In: Models, Algorithms, Logics and Tools. vol. 10460, pp. 466–483 (2017). https://doi.org/10.1007/978-3-319-63121-9_23
10. Chatzigiannakis, I., Michail, O., Spirakis, P.G.: Algorithmic verification of population protocols. In: Proc. 12th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS). pp. 221–235 (2010). https://doi.org/10.1007/978-3-642-16023-3_19
11. Clément, J., Delporte-Gallet, C., Fauconnier, H., Sighireanu, M.: Guidelines for the verification of population protocols. In: ICDCS. pp. 215–224. IEEE Computer Society (2011). https://doi.org/10.1109/ICDCS.2011.36
12. Cummings, R., Doty, D., Soloveichik, D.: Probability 1 computation with chemical reaction networks. Natural Computing **15**(2), 245–261 (2016). https://doi.org/10.1007/s11047-015-9501-x
13. Diamadi, Z., Fischer, M.J.: A simple game for the study of trust in distributed systems. Wuhan University Journal of Natural Sciences **6**(1), 72–82 (2001). https://doi.org/10.1007/BF03160228

14. Moran, P.A.P.: Random processes in genetics. Mathematical Proceedings of the Cambridge Philosophical Society **54**(1), 60–71 (1958). https://doi.org/10.1017/S0305004100033193
15. de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: Proc. 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 337–340 (2008). https://doi.org/10.1007/978-3-540-78800-3_24, z3 is available at https://github.com/Z3Prover/z3
16. Navlakha, S., Bar-Joseph, Z.: Distributed information processing in biological and computational systems. Communications of the ACM **58**(1), 94–102 (2014). https://doi.org/10.1145/2678280
17. Pang, J., Luo, Z., Deng, Y.: On automatic verification of self-stabilizing population protocols. In: Proc. 2nd IEEE/IFIP International Symposium on Theoretical Aspects of Software Engineering (TASE). pp. 185–192 (2008). https://doi.org/10.1109/TASE.2008.8
18. Schmidt, K.: LoLA: A low level analyser. In: Proc. 21st International Conference on Application and Theory of Petri Nets (ICATPN). pp. 465–474 (2000). https://doi.org/10.1007/3-540-44988-4_27, LoLA is available at http://service-technology.org/lola/
19. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Natural Computing **7**(4), 615–633 (2008). https://doi.org/10.1007/s11047-008-9067-y
20. Sun, J., Liu, Y., Dong, J.S., Pang, J.: PAT: Towards flexible verification under fairness. In: Proc. 21st International Conference on Computer Aided Verification (CAV). pp. 709–714 (2009). https://doi.org/10.1007/978-3-642-02658-4_59