

Black Ninjas in the Dark: Formal Analysis of Population Protocols

Michael Blondin
Fakultät für Informatik
Technical University of Munich
Garching bei München, Germany
blondin@in.tum.de

Stefan Jaax
Fakultät für Informatik
Technical University of Munich
Garching bei München, Germany
jaax@in.tum.de

Javier Esparza
Fakultät für Informatik
Technical University of Munich
Garching bei München, Germany
esparza@in.tum.de

Antonín Kučera
Faculty of Informatics
Masaryk University
Brno, Czech Republic
kucera@fi.muni.cz

Abstract

In this interactive paper, which you should preferably read connected to the Internet, the Black Ninjas introduce you to population protocols, a fundamental model of distributed computation, and to recent work by the authors and their colleagues on their automatic verification.

CCS Concepts • Theory of computation → Distributed computing models; Logic and verification;

Keywords population protocols, distributed computing, parameterized verification

ACM Reference Format:

Michael Blondin, Javier Esparza, Stefan Jaax, and Antonín Kučera. 2018. Black Ninjas in the Dark: Formal Analysis of Population Protocols. In *LICS '18: 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, July 9–12, 2018, Oxford, United Kingdom*. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3209108.3209110>

1 Black ninjas in the dark

“You can feel her in the dark.
She’s a black ninja!” — Battle Beast (2013)

The *Black Ninjas* are an ancient secret society of warriors. It is so secret that its members do not even know each other and how many they are. When there is a matter to discuss, Sensei, the founder of the society, asks the ninjas to meet at night, preferably during a storm as it minimizes the chance of being surprised by the enemy.

As it happens, all ninjas have just received a note asking them to meet in a certain Zen garden at midnight, wearing their black uniform, in order to decide whether they should attack a nearby castle at dawn. The decision is taken by majority, and in the case of a tie the ninjas will not attack. All ninjas must decide their vote

in advance, the only purpose of the meeting is to compute the final outcome.

When the ninjas reach the garden in the gloomy night, dark clouds cover the sky as rain pours vociferously. The weather is so dreadful that it is impossible to see or hear anything at all. For this reason, voting procedures based on visual or oral communication are hopeless. Is there a way for the ninjas to conduct their vote in spite of these adverse conditions?

1.1 A first protocol

Sensei has foreseen this situation and made preparations. The note sent to the ninjas contains detailed instructions on how to proceed. Each ninja must wander *randomly* around the garden. Two ninjas that happen to bump into each other exchange information using touch according to the following protocol.

Each ninja maintains two bits of information:

- the first bit indicates whether the ninja is currently *active* (A) or *passive* (P); and
- the second bit indicates the current *expectation* of each ninja on the final outcome of the vote: *yes, we will attack* (Y) or *no, we will not attack* (N).

This gives four possible states for each ninja: A_Y , A_N , P_Y , P_N . Initially the ninjas set their first bit to A, *i.e.*, they are all active, and their second bit to their vote. State changes obey *interaction rules* or *transitions* of the form $p, q \mapsto p', q'$, meaning that if the interacting ninjas are in states p and q , respectively, they move to states p' and q' . Sensei specifies two rules, shown on the left of Table 1, with the implicit assumption that for any combination of states not covered by the rules, the ninjas must simply keep their current states. The intuition behind the rules is as follows:

- The first rule asks two active ninjas with opposite votes to become passive and change their expectations to N. Intuitively, since the ninjas have opposite votes, they become passive to let the rest decide the outcome, and inform them of the final result. They move to N since, for all they know, they could be the only two ninjas of the society. In this case, there are no other ninjas and, since the vote is a tie, the final outcome is N.
- The second rule asks a passive ninja that bumps into an active ninja to change his or her expectation to the one of the active ninja. Strictly speaking, these are two rules, one for Y and one for N.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

LICS '18, July 9–12, 2018, Oxford, United Kingdom

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-5583-4/18/07...\$15.00

<https://doi.org/10.1145/3209108.3209110>

Rule Nr.	First protocol	Second protocol	Third protocol
1	$A_Y, A_N \mapsto P_N, P_N$	$A_Y, A_N \mapsto P_N, P_N$	$A_Y, A_T \mapsto A_Y, P_Y$
2	$A_\alpha, P_\beta \mapsto A_\alpha, P_\alpha \quad \alpha, \beta \in \{Y, N\}$	$A_\alpha, P_\beta \mapsto A_\alpha, P_\alpha \quad \alpha, \beta \in \{Y, N\}$	$A_Y, A_N \mapsto A_T, P_T$
3		$P_N, P_Y \mapsto P_N, P_N$	$A_T, A_N \mapsto A_N, P_N$
4			$A_T, A_T \mapsto A_T, P_T$
5			$A_\alpha, P_\beta \mapsto A_\alpha, P_\alpha \quad \alpha, \beta \in \{Y, T, N\}$

Table 1. Three protocols for computing majority.

Sensei reasons as follows: If initially there are more Y-ninjas than N-ninjas, then eventually all remaining active ninjas will be Y-ninjas (first rule), and they will eventually turn all passive ninjas into Y-ninjas (second rule). The opposite case is symmetric. So eventually all ninjas will have the same expectation, and will maintain it forever, *i.e.*, the ninjas will eventually reach a *stable consensus* and, further the value of the consensus (Y or N) will correspond to the correct outcome of the vote.

Throughout the paper, we will give links to interactive webpages. Each link will be depicted by a clickable pictogram of the form  which can alternatively be accessed as the n -th example found at <https://peregrine.model.in.tum.de/lics18>.

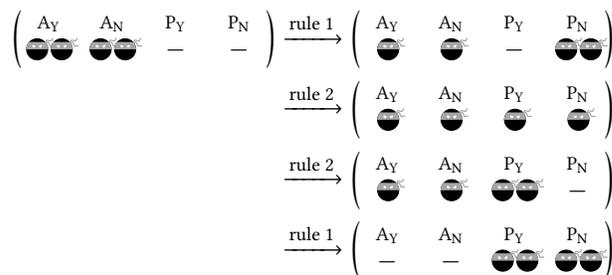
You can run the described protocol at . In the simulation, the ninjas are represented as circles wandering along the garden. States are represented by colours (see the legend on the screen). You can inspect the states and transitions of the protocol, and change the initial number of Y-ninjas and N-ninjas. The physical representation of the ninjas moving around is only for fun. In reality, the operational semantics of the protocol is as follows: at each moment in time, a pair of ninjas is picked uniformly at random, and the corresponding rule (if any) is applied. By clicking the menu tab *Population View*, you can see another visualization showing only the number of ninjas in each state evolving over time.

Observe that no individual ninja can know with certainty that a consensus has already been reached. Indeed, at each moment in time, a ninja has interacted only with a finite number of other ninjas, and, since the ninjas do not know how many they are, they cannot exclude the existence of many other individuals that could flip the current outcome of the vote. However, this does not make the protocol useless. Indeed, from the size of the garden, the ninjas can deduce an upper bound for the number of ninjas, and from this upper bound and the mean time between interactions they can compute bounds on the probability to have reached consensus after a given time. In particular, a computation of this kind (or a sufficiently large number of simulations) can ensure that consensus is reached before dawn with large probability.

1.2 A second protocol

The protocol works fine for a time, but then disaster strikes. At one gathering there is an equal number of Y-ninjas and N-ninjas. In this case – and only in this case – the protocol is incorrect, and the ninjas may never reach a consensus. In fact, this happens with large probability. You can experiment with this scenario by clicking again the link above, and changing the initial number of ninjas to, say, five ninjas each in states A_Y and A_N .

Consider the case in which there are initially four ninjas, two in state A_Y , and two in state A_N . One possible execution is:



after which the states of the ninjas cannot change anymore. Only half of the ninjas attack at dawn, they are massacred, and Sensei commits harakiri. Very sad story.

It is not difficult to see what goes wrong. If the first rule had priority over the second rule, then the protocol would indeed work correctly even in the case of a tie. The active ninjas would become passive in pairs, after which an N-consensus would be reached, which corresponds to the right outcome (“in the case of a tie, do not attack”). However, in the actual protocol, the second rule “interferes” with the first rule. For example, consider an execution in which the active ninjas become passive in pairs by means of the first rule, until only one active pair A_Y, A_N remains. The remaining A_Y -ninja may now turn arbitrarily many ninjas to Y (interference of the second rule), and only then interact with the A_N ninja (first rule again). After this we have arbitrary numbers of Y and N ninjas, and no rule is applicable anymore.

The newly elected Sensei II analyzes the problem and quickly comes up with a repair for the protocol. It is shown in the middle of Table 1: a new rule $P_Y, P_N \mapsto P_N, P_N$ is added. If all ninjas become passive, which can only happen in the case of a tie, then the new rule guarantees that an N-consensus is eventually reached. This new protocol can be simulated at .

1.3 A third protocol

Again, the new protocol works fine ... until it doesn't. The story repeats itself: At dawn no consensus has been reached, only some ninjas attack, they are decimated, Sensei II commits harakiri ... In a farewell note he describes his puzzlement:

I have studied Math with the best Zen masters and I have carefully crafted a mathematical proof showing that the protocol eventually reaches a consensus with probability 1. But the heavens did not bless my efforts ... it was not enough. Unable to complete this heavy task, arrows all spent, so sad I fall.

The successor, Sensei III, is well versed in new technologies and quickly writes a program to simulate the behaviour of the protocol for different initial numbers of ninjas. You can conduct simulations of the second protocol together with statistics at [3].

In each simulation, the number of ninjas is chosen uniformly at random between 10 and 15, and the simulation runs for 500 steps, or until no rule is applicable. We see that the protocol does not reach a consensus in about 25% of the runs on average, and scrolling down the page we see that this happens only for the case in which the initial numbers of Y-ninjas is slightly above the number of N-ninjas.

Sensei III considers then the general scenario in which Y has a majority of only one ninja, and finds the following explanation: In this situation, the protocol reaches with high probability a configuration with one single ninja in state A_Y and many ninjas in states P_N . There is now a struggle between the single A_Y ninja, who turns P_N -ninjas to P_Y using the second rule, against the many P_N -ninjas, who turn P_Y -ninjas back to P_N using the new rule. The A_Y -ninja eventually “wins”, and consensus Y is reached, but only after she turns *all* P_N -ninjas to P_Y *before* any of the P_N -ninjas converts any of them back to P_Y . As we will see in Section 5, for an initial configuration with 8 Y-ninjas and 7 N-ninjas, this takes on average 9,072,494 steps, and dawn arrives long before consensus!

Sensei III wants a new protocol with a clean design. Since ties are the source of all problems, she decides that the protocol should explicitly deal with them. So, apart from being active or passive, ninjas can now have a more refined expectation of the outcome: Y, N, and T (for “tie”). The protocol is shown on the right of Table 1. When two active ninjas meet, only one of them becomes passive, and both change their expectation in the natural way. For example, if the expectations are Y and T, then the ninja with expectation T changes it to Y. This explains rules 1 to 4. Rule 5 is the usual one: passive ninjas adopt the expectation of active ninjas. You can simulate the protocol at [4]. In simulations with up to 15 ninjas, consensus never takes more than 170 steps: [5].

1.4 Sensei III's questions

The new protocol is working fine but, given the fate of her two predecessors, Sensei III does not feel completely at rest ... Perhaps some formal methods could contribute to her peace of mind? She has several obvious questions:

- What is a protocol?
- What does it mean for a protocol to be correct? What does it mean for it to be efficient?
- How can I ensure that a protocol is correct and efficient?

Moreover, apart from these verification questions, she is also interested in other problems:

- Are there protocols for other tasks? For example, if the ninjas should attack only if at least 90% of them vote for it, could the protocol be changed? And what if 4 votes or more can veto the attack?
- Some ninjas cannot remember complicated protocols. What is the minimal number of states of a protocol for a given task?

In 2004, Sensei III discussed these questions with a team of computer scientists at Yale University. The latter formally defined the *population protocol* model [3], and started answering some of the questions of Sensei III. Other researchers then joined in. In the rest

of the paper, we review some of the results they found, and devote special attention to questions (c) and (e) studied by the authors.

2 What is a (correct) protocol?

The ninja majority protocols of Section 1 can all be seen as population protocols, an elegant formalism, introduced by Angluin *et al.*, for distributed systems of mobile agents with little computational power [3]. Such protocols allow for modeling systems such as networks of passively mobile sensors and chemical reaction networks.

Formally, a *population protocol* is a tuple $\mathcal{P} = (Q, T, I, O)$ such that

- Q is a finite set of elements called *states*,
- $T \subseteq Q^2 \times Q^2$ is a finite set of rules called *transitions*,
- $I \subseteq Q$ is a subset of states said to be *initial*,
- $O: Q \rightarrow \{0, 1\}$ maps each state to an *output*.

For simplicity, we denote a transition $((p, q), (p', q')) \in T$ as $p, q \mapsto p', q'$, and we assume T contains transition $p, q \mapsto p, q$ for every $p, q \in Q$.

A *configuration* is a vector $C \in \mathbb{N}^Q$ assigning to each state q a number $C(q)$ of agents (*i.e.* ninjas) in state q . A configuration is *initial* if $C(q) = 0$ for every $q \notin I$. For example, $(1, 1, 0, 2)$ represents the following pictorial configuration of ninjas described in Section 1.2:

$$\left(\begin{array}{cccc} A_Y & A_N & P_Y & P_N \\ \bullet & \bullet & - & \bullet \bullet \end{array} \right).$$

We write $C \xrightarrow{t} D$ for a transition $t: (p, q) \mapsto (p', q')$ if changing an agent from state p to p' and another agent from state q to q' in configuration C results in configuration D . We write $C \xrightarrow{*} D$ if D can be reached from C through a (possibly empty) sequence of transitions. A configuration C is *terminal* if $C \xrightarrow{*} D$ implies $D = C$, *i.e.* C cannot be changed. For example, in the first protocol of Table 1, we have $(2, 2, 0, 0) \xrightarrow{*} (0, 0, 2, 2)$ where the latter configuration is terminal.

An execution is an infinite sequence of configurations C_0, C_1, \dots for which there exist transitions $t_1, t_2, \dots \in T$ such that

$$C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} C_2 \xrightarrow{t_3} \dots$$

An execution is said to be *fair* if for every configuration D , if $\{i \in \mathbb{N} : C_i \xrightarrow{*} D\}$ is infinite, then $\{i \in \mathbb{N} : C_i = D\}$ is infinite. In other words, a fair execution cannot avoid a configuration reachable infinitely often. Fairness is an abstraction of the interactions occurring stochastically among agents.

The *output* of a configuration C is defined as

$$O(C) = \begin{cases} b & \text{if } O(q) = b \text{ for every } q \in Q, \\ \perp & \text{otherwise.} \end{cases}$$

A configuration C is said to be *stable* if $O(C) \in \{0, 1\}$ and for every configuration D the following holds: if $C \xrightarrow{*} D$, then $O(D) = O(C)$. In other words, a configuration is stable if it has reached a consensus which cannot be reverted.

A protocol \mathcal{P} *computes* a predicate $\varphi: \mathbb{N}^I \rightarrow \{0, 1\}$ if for every initial configuration C , every fair execution from C leads to a stable configuration D such that $O(D) = \varphi(C)$. We say that a protocol \mathcal{P} is *correct* with respect to a predicate φ if \mathcal{P} computes φ . The three protocols of Section 1, under outputs

$$O(A_Y) = O(P_Y) = 1 \text{ and } O(A_N) = O(P_N) = O(A_T) = O(P_T) = 0,$$

attempt to compute the predicate φ such that $\varphi(C) = 1 \iff C(A_Y) > C(A_N)$. Only the second and third protocols are correct.

For a survey on population protocols and their variants which will not be covered here, see the recent review article of Michail and Spirakis [22] on the theory of dynamic networks, or the introduction to population protocols by Aspnes and Ruppert [7].

3 What predicates can ninjas compute?

Sensei III's fourth question was given a precise answer by Angluin *et al.* [3, 4, 6]: Population protocols can compute precisely all Presburger-definable predicates [3, 4]. Recall that a predicate is *Presburger-definable* if it can be defined in the first-order logic over \mathbb{N} with addition and order, *i.e.* in $\text{FO}(\mathbb{N}, +, <)$. The proof that population protocols can compute all Presburger predicates is not very complicated, once one observes that Presburger arithmetic admits a quantifier elimination procedure [25] (see [12], *e.g.*, for a modern presentation). By this result, every Presburger-definable predicate is equivalent to a boolean combination of *threshold* and *modulo* predicates respectively of the form

$$\sum_{i=1}^n a_i x_i \leq b \text{ and } \sum_{i=1}^n a_i x_i \equiv b \pmod{c}.$$

It is not so difficult to give families of protocols for these predicates. You can inspect and simulate a protocol for $2x - y \leq 1$ at [6], and one for $x \equiv 2 \pmod{3}$ at [7]. Now, it only remains to show that the class of predicates computed by protocols is closed under boolean operations. This is achieved by means of constructions closely resembling those for finite automata. Indeed, flipping the outcomes of the states of a protocol \mathcal{P} computing φ yields a protocol \mathcal{P}' computing $\neg\varphi$, *i.e.*, we replace the mapping $O: Q \rightarrow \{0, 1\}$ by the mapping \bar{O} given by $\bar{O}(q) = 1 - O(q)$ for every $q \in Q$. For conjunction, there is also a simple product construction. Given protocols $\mathcal{P}_1 = (Q_1, T_1, I_1, O_1)$ and $\mathcal{P}_2 = (Q_2, T_2, I_2, O_2)$ computing respectively predicates φ_1 and φ_2 , the protocol $\mathcal{P}_1 \times \mathcal{P}_2$ with states $Q_1 \times Q_2$, outputs defined by $O((q_1, q_2)) = O_1(q_1) \wedge O_2(q_2)$, and transitions defined in the expected way, computes $\varphi_1 \wedge \varphi_2$ (see, *e.g.*, [3, Sect. 4] for details).

The proof that population protocols can *only* compute Presburger predicates is much harder. Currently there exist two independent proofs. The original proof by Angluin *et al.* [4, 6] applies results from convex geometry. The second proof by Esparza *et al.* [17] is based on the theory of vector addition systems, and leads to an algorithm to construct, given a protocol, a representation of the predicate it computes (if any) as a Presburger formula or as a semilinear set.

4 Are the senseis' protocols correct?

Once a population protocol \mathcal{P} has been designed for computing a certain predicate φ , it is natural to ask whether \mathcal{P} indeed computes φ correctly. For example, how can we be sure that the protocols designed by the senseis in Section 1 really compute the predicate $\#Y > \#N$?

4.1 Handcrafted proofs

One possible way is to manually establish some sorts of invariants and progress measures in order to derive a formal proof. Let us illustrate this idea on the second protocol of Table 1. Observe that for every configurations C and D such that $C \xrightarrow{t} D$ for some transition

$t \in T$, the following holds:

$$C(A_Y) + C(A_N) \geq D(A_Y) + D(A_N). \quad (1)$$

$$C(A_Y) - C(A_N) = D(A_Y) - D(A_N), \quad (2)$$

$$D(A_Y) + D(A_N) + D(P_N) > 0. \quad (3)$$

By inequality (1), the number of active agents in any fair execution must eventually stabilize, and hence at most one type of active agents, namely A_Y or A_N , eventually remains. If some active agent A_α remains, then, by rule 2 combined with fairness, all passive agents will be converted to P_α , leading to a stable terminal configuration. Invariant (2) ensures that the reached consensus is correct.

In the other case where no active agent remains, invariant (3) ensures that some agent in state P_N remains and, again by fairness and (2), it will also convert all passive agents to the correct consensus.

Coming up with such invariants and progress measures becomes tricky as protocols get more complex. Moreover, there is a serious risk of missing some corner cases or of introducing subtle bugs upon implementing a protocol. As an example, we challenge the reader to find the typo in the more complex majority protocol depicted in Figure 1!

4.2 Towards an automatic approach

One way to alleviate some of the issues just discussed is to get assistance from an interactive theorem prover, *e.g.* Deng and Monin successfully verified a simple leader election protocol with Coq [16]. While more reliable, this approach further increases the burden as protocols get more complicated.

One alternative way to gain additional confidence in the correctness of a protocol consists in fixing an upper bound n on the number of agents. Off-the-shelf or custom model checking tools can then be used to verify whether a protocol is correct for configurations of size up to n . This approach has been studied and implemented by various authors [13, 14, 24, 27]. While fully automatic, such “incomplete algorithms” do not prove correctness. Moreover, due to state-space explosion, n is typically limited to small numbers.

Determining algorithmically whether correctness holds is a challenging task since it involves reasoning about *infinitely many* initial configurations, *i.e.* every n . In fact, it is *a priori* not even clear whether this is a decidable decision problem.

By an intricate reduction to the Petri net reachability problem, Esparza, Ganty, Leroux and Majumdar have recently shown that verifying the correctness of a population protocol is decidable [17]. This result has opened the door for the design and implementation of complete algorithms for the automatic verification of population protocols. Unfortunately, the current best upper bound on the complexity of the Petri net reachability problem is hyper-Ackermanniann [20].

4.3 A truly automatic approach

Formally, determining whether a population protocol \mathcal{P} computes a predicate φ amounts to testing whether the following formula is *unsatisfiable*:

$$\exists C, D : C \xrightarrow{*} D \quad (4)$$

$$\wedge C \text{ is initial} \quad (5)$$

$$\wedge D \text{ is bottom} \quad (6)$$

$$\wedge O(D) \neq \varphi(C) \quad (7)$$

Parameters:
 m , an odd integer > 0 , and d , integer > 0
Input: States of two nodes, x and y
Output: Updated states x' and y'

```

1  $weight(x) = \begin{cases} |x| & \text{if } x \in \text{StrongStates or } x \in \text{WeakStates;} \\ 1 & \text{if } x \in \text{IntermediateStates.} \end{cases}$ 
2  $sgn(x) = \begin{cases} 1 & \text{if } x \in \{+0, 1_d, \dots, 1_1, 3, 5, \dots, m\}; \\ -1 & \text{otherwise.} \end{cases}$ 
3  $value(x) = sgn(x) \cdot weight(x)$ 
4  $\phi(x) = -1$  if  $x = -1$ ;  $1$  if  $x = 1$ ;  $x$ , otherwise
5  $R_\downarrow(k) = \phi(k)$  if  $k$  odd integer,  $k - 1$  if  $k$  even)
6  $R_\uparrow(k) = \phi(k)$  if  $k$  odd integer,  $k + 1$  if  $k$  even)
7  $Shift\text{-}to\text{-}Zero(x) = \begin{cases} -1_{j+1} & \text{if } x = -1_j \text{ for some index } j < d \\ 1_{j+1} & \text{if } x = 1_j \text{ for some index } j < d \\ x & \text{otherwise.} \end{cases}$ 
8  $Sign\text{-}to\text{-}Zero(x) = \begin{cases} +0 & \text{if } sgn(x) > 0 \\ -0 & \text{otherwise.} \end{cases}$ 

```

State Space:
 $\text{StrongStates} = \{-m, -m + 2, \dots, -3, 3, 5, \dots, m - 2, m\}$,
 $\text{IntermediateStates} = \{-1, -1_2, \dots, -1_d, 1_d, 1_{d-1}, \dots, 1_1\}$,
 $\text{WeakStates} = \{-0, +0\}$

```

9 procedure update( $x, y$ )
10 if ( $weight(x) > 0$  and  $weight(y) > 1$ ) or ( $weight(y) > 0$  and  $weight(x) > 1$ ) then
11  $x' \leftarrow R_\downarrow\left(\frac{value(x)+value(y)}{2}\right)$  and  $y' \leftarrow R_\uparrow\left(\frac{value(x)+value(y)}{2}\right)$ 
12 else if  $weight(x) \cdot weight(y) = 0$  and  $value(x) + value(y) > 0$  then
13 if  $weight(x) \neq 0$  then  $x' \leftarrow Shift\text{-}to\text{-}Zero(x)$  and  $y' \leftarrow Sign\text{-}to\text{-}Zero(x)$ 
14 else  $y' \leftarrow Shift\text{-}to\text{-}Zero(y)$  and  $x' \leftarrow Sign\text{-}to\text{-}Zero(y)$ 
15 else if ( $x \in \{-1_d, +1_d\}$  and  $weight(y) = 1$  and  $sgn(x) \neq sgn(y)$ ) or
16 ( $y \in \{-1_d, +1_d\}$  and  $weight(x) = 1$  and  $sgn(y) \neq sgn(x)$ ) then
17  $x' \leftarrow -0$  and  $y' \leftarrow +0$ 
18 else
19  $x' \leftarrow Shift\text{-}to\text{-}Zero(x)$  and  $y' \leftarrow Shift\text{-}to\text{-}Zero(y)$ 

```

Figure 1. Average-and-conquer majority protocol of [2]. A typo¹ appears in the original figure of [2] which we reproduce here.

where “ D is bottom” means that every configuration reachable from D can reach it back, i.e. $D \xrightarrow{*} D' \iff D' \xrightarrow{*} D$ for every configuration D' .

Therefore, one line of attack for verifying correctness consists in using an SMT solver to test whether the above formula is unsatisfiable. Conjuncts (5) and (7) can be encoded respectively as the two following Presburger formulas:

$$\bigwedge_{q \in Q \setminus I} C(q) = 0, \\ \bigvee_{q \in Q} (D(q) > 0 \wedge O(q) \neq \varphi(C)).$$

However, it is not clear how to encode conjunct (6), and encoding conjunct (4) is simply impossible since the reachability relation of a population protocol is not Presburger-definable in general.

Fortunately, there is a way to circumvent these two issues [10]. First, the reachability relation in conjunct (4) can be replaced by a suitable over-approximation. Secondly, most protocols found in the literature are *silent*: all fair executions lead to terminal configurations. Therefore, assuming that a protocol is silent, one can replace conjunct (6) by “ D is terminal”, which is Presburger-definable:

$$\bigwedge_{\substack{\text{rule } p, q \rightarrow p', q' \\ \text{s.t. } \{p, q\} \neq \{p', q'\}}} [(p \neq q \rightarrow (D(p) = 0 \vee D(q) = 0)) \wedge (p = q \rightarrow (D(p) \leq 1))].$$

4.4 How to make it work?

For the approach just described to work, two crucial questions must be answered:

1. What is a good over-approximation of reachability?
2. How can we be sure that a protocol is silent?

As it turns out, the first question can be answered by borrowing an over-approximation from Petri net theory that has been recently applied in other contexts: the so-called state-equation + trap/siphon constraints (e.g., see [18]).

Here, we focus our attention on the answer to the second question (see [10, Sect. 4.1] for the detailed answer to question 1). Let us consider again the second protocol of Table 1. We have already seen that this protocol is silent as all fair executions end up in terminal configurations. In fact, the protocol’s structure is richer: it works in *layers*. First, the active agents “collide” until at most one type

of active agents remains. Then, either agents in state A_Y convert all agents in state P_N , or agents in states A_N and P_N convert all agents in state P_Y . These layers are illustrated as a directed tree in Figure 2.

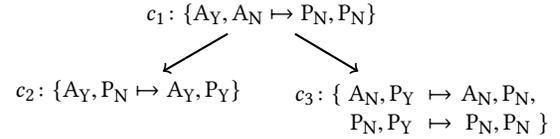


Figure 2. Layer graph of the second protocol of Table 1.

For every node c of Figure 2, let T_c be the transitions labeling node c . Let us fix some node c . Any (fair or unfair) execution of the protocol restricted to the transitions of T_c must be finite. Moreover, once transitions T_c are disabled, executing transitions T_d , of some successor node d , cannot lead to a configuration in which a transition of T_c becomes enabled. In short, every subprotocol induced by a node of the graph is silent and cannot be re-enabled upon termination.

Such a layer graph is a certificate of silentness since:

- from any initial configuration, it is possible to reach a terminal configuration by repeatedly executing transitions of a node and moving down the layer graph upon termination of the node;
- by fairness, any initial configuration must eventually reach such a terminal configuration.

It is crucial to see that an execution needs not to follow the structure of the layer graph. The latter only *proves* silentness of the protocol.

As observed by the authors, most protocols found in the literature admit a layer graph, or equivalently in the terminology of [10]: satisfy *layered termination*. In fact, most protocols are naturally designed with such layers in mind. Moreover, the problem of determining whether a layer graph exists belongs to NP and can be expressed as a Presburger formula that can be fed to an SMT solver, thus answering Question 2.

5 Will ninjas agree before dawn?

Now that the Sensei knows how to determine whether her plans are sound, she wants to know how to determine whether the ninjas will agree before dawn, or more formally: whether the expected

¹ $0 \neq (\hat{n})\hat{n}l\hat{n}l\hat{n}l + (x)\hat{n}l\hat{n}l\hat{n}l \text{ and } 0 < (\hat{n})\hat{n}l\hat{n}l\hat{n}l + (x)\hat{n}l\hat{n}l\hat{n}l$

running time of a given protocol is fast or not in terms on the number of ninjas.

5.1 Expected convergence time

While fairness allows for reasoning about predicates computed by population protocols, it is not a sufficient assumption for analyzing convergence time. For this reason, it is often assumed that at each step of an execution, a pair of agents is selected uniformly at random. The expressive power of the resulting model is unchanged: a protocol computes a predicate φ with probability 1 if and only if it computes φ under the fairness assumption.

In the probabilistic setting, we define the (*sequential*) *expected convergence time* of a protocol \mathcal{P} as the mapping $t_{\mathcal{P}}(n)$ assigning to every $n \geq 2$, the maximal expected number of steps required, over initial configurations of size n , to reach a stable consensus. Intuitively, this corresponds to a sequential model in which all interactions occur one after another. The *parallel expected convergence time* is typically defined as $t_{\mathcal{P}}(n)/n$. It corresponds to an asynchronous model in which every agent has an internal clock ticking at the times of a rate 1 Poisson process [11]. When the clock of agent A ticks, another agent B is selected uniformly at random, and A and B interact. For n agents, there are n independent clocks, and the time between two consecutive interactions has exponential distribution with rate n . Hence, the expected number of interactions in one time unit is n .

It was shown by Angluin *et al.* [3] that every Presburger-definable predicate can be computed by a protocol whose expected convergence time is in $O(n^2 \log n)$. This bound can be improved to $O(n \log^5 n)$ by means of a *protocol with a leader* [5]. (We delay the introduction of protocols with leaders to the next section.) More recent work on the topic is outside the scope of this paper (see, e.g., [1, 2]).

5.2 Towards an automatic approach

It is possible to obtain some knowledge on the expected convergence time of a population protocol by analyzing the case of a fixed population size n . This can be done by simulating a protocol from randomly chosen initial configurations [9], or by using a probabilistic model checker [14, 23]. For example, one can translate a population protocol as a Markov chain, for a fixed population size n , and compute the expected number of steps before reaching a stable configuration. The results obtained using the tool PRISM [19] for the three protocols of Table 1, with $n = 15$, is illustrated in Figure 3. In particular, we observe that the second protocol appears to be extremely slow in the case of a slight majority of A_Y , while the two other protocols seem to be reasonably fast overall.

5.3 A truly automatic approach

As in the case of correctness, the approach just described does not give any formal guarantee on expected convergence times. For this reason, we are currently developing an alternative to the layer graphs of Section 4 in order to *automatically* derive *asymptotic* upper bounds on expected convergence times. These alternative *stage graphs* store sets of configurations via a simple temporal logic

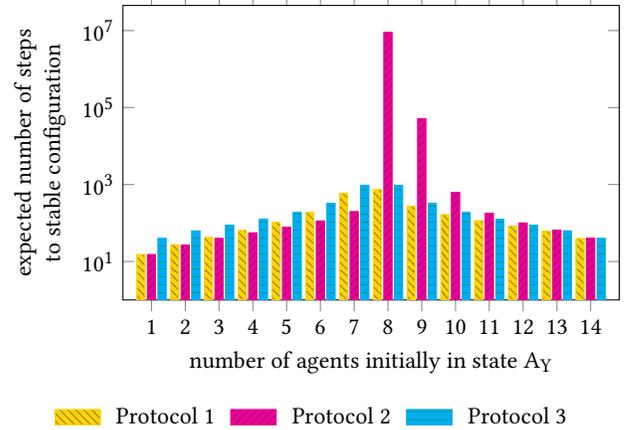


Figure 3. Expected number of steps before reaching a stable configuration, for each protocol of Table 1 starting from initial configurations of size 15.

whose syntax and semantics are defined as follows:

$$\begin{aligned}
 C \models q &\iff C(q) \geq 1, \\
 C \models q! &\iff C(q) = 1, \\
 C \models \varphi \wedge \psi &\iff C \models \varphi \text{ and } C \models \psi, \\
 C \models \neg\varphi &\iff C \not\models \varphi, \\
 C \models \Box\varphi &\iff D \models \varphi \text{ for every } D \text{ reachable from } C,
 \end{aligned}$$

where C, D denote configurations and q a state.

A *stage graph* is a directed acyclic graph $(\mathbb{S}, \rightarrow)$ such that

- every node $S \in \mathbb{S}$ is associated to a formula φ_S and a set of configurations $\llbracket S \rrbracket = \{C : C \models \varphi_S\}$;
- for every initial configuration C , there exists $S \in \mathbb{S}$ such that $C \in \llbracket S \rrbracket$;
- for every $S \in \mathbb{S}$ and $C \in \llbracket S \rrbracket$, any fair execution from C leads to some $C' \in \llbracket S' \rrbracket$ such that $S \rightarrow S'$, i.e. C leads to a successor configuration with probability 1.

Let us fix a stage graph $G = (\mathbb{S}, \rightarrow)$ for some protocol \mathcal{P} . We say that a node $S \in \mathbb{S}$ is *stable* if all configurations of $\llbracket S \rrbracket$ are stable. If all bottom nodes of G are stable, then G can be used to derive an upper bound on the expected convergence time of \mathcal{P} . Indeed, any execution of \mathcal{P} must start from some node S and end up in some stable bottom node S' . Therefore, an upper bound on $t_{\mathcal{P}}$ can be obtained by bounding the expected time to move from a node of G to its successors.

5.4 Stage graphs: an example

As discussed in Section 1, the third majority protocol of Table 1 does not suffer from the slow convergence time of the second protocol. This can be derived from the stage graph $G = (\mathbb{S}, \rightarrow)$ depicted in Figure 4.

Let us consider a fair execution C_0, C_1, \dots from some initial configuration C_0 . Note that $C_0 \in \llbracket S_0 \rrbracket$. If C_0 consists only of state A_Y or only of state A_N , then C_0 is stable, and hence the execution has already stabilized. This scenario corresponds to the edges from S_0 to S_1 and S_2 .

If $C_0(A_Y) \geq 1$ and $C_0(A_N) \geq 1$, then rules 1 to 3 eventually become permanently disabled after some i steps. Moreover, when

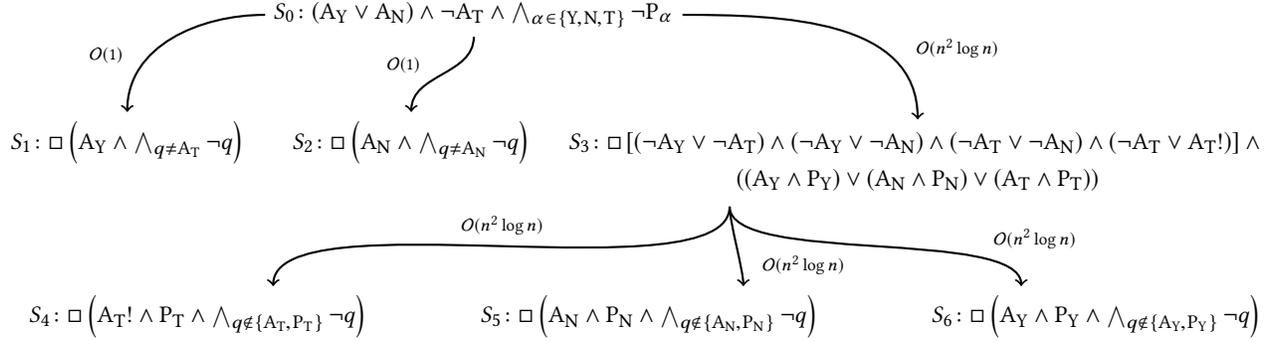


Figure 4. A stage graph for the third majority protocol of Table 1 where each state S_i is labeled by its formula φ_{S_i} .

this happens, it is the case that $C_i(A_\alpha) \geq 1$ and $C_i(P_\alpha) \geq 1$ for some $\alpha \in \{Y, N, T\}$. This scenario corresponds to the edge from S_0 to S_3 . Rules 4 and 5 also become eventually disabled, leading to a terminal configuration C_j consisting only of states A_α and P_α . This scenario corresponds to the edges from S_3 to S_4 , S_5 and S_6 .

It can be shown that moving from each node of G to one of its successors takes an expected time of $O(n^2 \log n)$. Thus, the expected convergence time of the protocol is in $O(n^2 \log n)$. Moreover, with the machinery currently developed by some of the authors, G and these bounds can be derived automatically in less than a second.

6 How much memory do ninjas need?

The Sensei organizes a more elaborated castle attack that requires at least 9 ninjas to be successful. She needs a protocol for the predicate $x \geq 9$.

6.1 A protocol for $x \geq c$ with $O(c)$ states

Confident in her new skills at devising protocols, the sensei quickly crafts one for the task. Each ninja carries a purse initially containing one pebble. When two ninjas meet, one of them gives all of his pebbles to the other. If a ninja collects 9 pebbles, he tells the others that they have reached the necessary number and asks them to spread the news. More precisely:

- the states are $\{0, 1, \dots, 9\}$, with $O(9) = 1$ and $O(x) = 0$ for $x \neq 9$; initially all ninjas are in state 1;
- the transitions are $x, y \mapsto 0, \min(9, x + y)$ and $9, x \mapsto 9, 9$ for every states x, y .

You can run the protocol at [\[8\]](#). It is easy to see that the protocol works not only for 9, but for any constant $c \geq 1$. More precisely, we have a family of protocols of $c + 1$ states for the family of predicates $x \geq c$.

6.2 A protocol for $x \geq c$ with $O(\log c)$ states

The protocol above has ten states. The sensei is not satisfied with this as she is convinced ninjas do not have to carry this many pebbles. After some time, she comes up with a protocol with only six states. The states are $\{0, 1, 2, 4, 8, 9\}$, with $O(9) = 1$ and $O(x) = 0$ for $x \neq 9$; and initially all ninjas are in state 1. The transitions are

- $x, x \mapsto 2x, 0$ and $2x, 0 \mapsto x, x$ for every $x \in \{1, 2, 4\}$;
- $8, 1 \mapsto 9, 9$; and
- $9, x \mapsto 9, 9$ for every $x \in \{0, 1, 2, 4, 8\}$.

You can run the protocol at [\[9\]](#).

It requires more work to generalize the idea of this protocol to produce a family of protocols for every constant c . The set of states seems easy to generalize: For $x \geq c$, it should contain $\{0, 2^0, 2^1, \dots, 2^{\lceil \log c \rceil}\}$. However, getting the transitions right requires some additional states, and quite some care². Nonetheless it can be done, and for every $c \geq 0$ we obtain a protocol with $\Theta(\log c)$ states that computes predicate $x \geq c$. The details of the construction can be found in [8].

Is $\Theta(\log c)$ optimal? A simple counting argument yields:

Theorem 6.1 ([8]). *There exist infinitely many $c \geq 1$ such that every protocol computing $x \geq c$ has at least $(\log c)^{1/4}$ states.*

So, for example, there can be no family of predicates computing $x \geq c$ with $O(\log \log c)$ states for every c . But is this possible for some c ? To the best of our knowledge, this question is open for population protocols as described in Section 2. However, the answer is positive for the slightly more general class of *population protocols with a constant number of leaders*.

6.3 Protocols with leaders

Intuitively, leaders are auxiliary agents whose number is finite and potentially known: Think of a finite number of red ninjas that do not belong to the society, but help to conduct the protocols. A protocol with leaders has an additional set Q_ℓ of leader states, disjoint from the set Q of regular states, containing a distinguished initial state $q_{\ell 0}$, and an additional set of rules of the form $(q_\ell, q) \mapsto (q'_\ell, q')$, where $q_\ell, q'_\ell \in Q_\ell$ and $q, q' \in Q$. The number of leaders is fixed by the set of initial configurations: If the protocol has k leaders, then a configuration is initial if and only if it has exactly k agents in $q_{\ell 0}$ (the leaders), and an arbitrary number of agents in the initial regular states. One can also have different sets $Q_{\ell_1}, Q_{\ell_2}, \dots, Q_{\ell_m}$ of leader states (think of red, yellow, green ninjas ...), with initially k_1, k_2, \dots, k_m agents.

Protocols with a constant number of leaders have the same expressive power as leaderless protocols. However, with the help of leaders one can design simpler, faster, or more compact protocols. Imagine Sensei III has decided to change the voting rules: The ninjas will only attack if Y wins by at least 4 votes. It is easy to design a new protocol with 4 red ninjas. Intuitively, the red ninjas pretend to vote N, which reduces the problem to the previous protocol for

²To convince yourself of this, we suggest you try to design the protocol for $c = 7$ or, more generally, for constants of the form $c = 2^d - 1$.

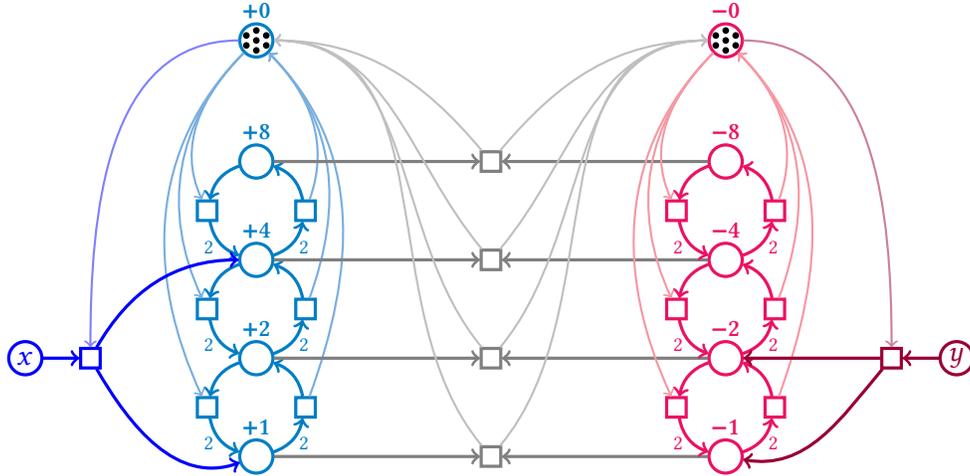


Figure 5. Petri net representation of a population protocol with 14 leaders computing $5x - 3y > 0$.

simple majority. More precisely, any protocol computing the predicate $x \geq y$ can be transformed into a protocol computing $x \geq y + 4$ as follows:

- Split each state $q \in Q$ into two states $q^b \in Q^b$ and $q^r \in Q^r$ (b for “black” and r for “red”), with the same output as q .
- Split each transition $q_1, q_2 \mapsto q_3, q_4$ into 16 transitions $q_1^{c_1}, q_2^{c_2} \mapsto q_3^{c_3}, q_4^{c_4}$ for $c_1, \dots, c_4 \in \{r, b\}$;
- if q_x, q_y are the initial states for the variables x and y , then let q_x^b and q_y^b be the initial states for the black ninjas, and q_x^r the initial state for the red ninjas. (Many of these transitions may not be necessary.)

The protocol obtained by applying this transformation to Sensei III’s protocol can be simulated at [10](#). Some “useless” transitions have been removed.

You may try to design a “leaderless” protocol for predicate $x \geq y + 4$ or, alternatively, inspect and simulate one at [11](#).

6.4 Some thresholds have very succinct protocols

Let us go back to the question stated before our excursion into protocols with leaders: Is there a family of predicates $x \geq c$ having protocols of size $O(\log \log c)$? The answer is positive for protocols with two leaders:

Theorem 6.2 ([8]). *There exists a family $\{\mathcal{P}_0, \mathcal{P}_1, \dots\}$ of protocols with two leaders and a family $\{c_0, c_1, \dots\}$ of natural numbers such that for every $n \in \mathbb{N}$, the protocol \mathcal{P}_n has $O(\log \log c_n)$ states and computes $x \geq c_n$.*

The proof of the theorem relies on a result by Mayr and Meyer on succinct representations of commutative semigroups [21]. Whether the same bound can be achieved with “leaderless” protocols is an open question, and so is determining whether the $\log \log c$ bound is tight. However, [8] provides some partial information.

We mentioned in Section 1 that ninjas executing a correct majority protocol — like the protocols of Sensei II and Sensei III — will eventually reach a consensus, but they never have absolute certainty that the consensus will be 0 or 1, even after reaching it! However, for ninjas executing the $O(\log c)$ protocol for $x \geq c$, the situation is different: when they converge to 1, they eventually

know with certainty that they will converge to 1. Indeed, in this case all ninjas eventually reach state c , and once they do, they know the consensus will be 1. A protocol with this property is called *1-aware*. It is not difficult to show that a predicate $\varphi(x_1, x_2, \dots, x_n)$ can be computed by a 1-aware protocol if and only if it is monotone, meaning that $\varphi(x_1, x_2, \dots, x_n) = 1$ and $\bigwedge_{i=1}^n y_i \geq x_i$ implies $\varphi(y_1, y_2, \dots, y_n) = 1$.

Theorem 6.3 ([8]). *Every 1-aware, leaderless population protocol computing $x \geq c$ has at least $\Omega(\log c)$ states. Every 1-aware population protocol with a constant number of leaders computing $x \geq c$ has at least $\Omega((\log \log c)^{1/9})$ states.*

6.5 Succinct protocols for linear inequalities

The upper bound of Section 6.2 can be generalized if leaders are allowed: any predicate of the form

$$\bigwedge_{i=1}^m a_{i,1}x_1 + a_{i,2}x_2 + \dots + a_{i,n}x_n > c_i$$

can be computed by a protocol with $14m(\log m + k)$ leaders and $27(\log m + k)(m + n)$ states, where k is the number of bits in the binary representation of

$$\max\{|a_{i,1}|, |a_{i,2}|, \dots, |a_{i,n}|, |c_i| : 1 \leq i \leq m\}.$$

To demonstrate how this can be achieved, we give a sketch of the protocol $\mathcal{P} = (Q, T, I, O)$ for the predicate $5x - 3y > 0$. The states are defined as $Q = Q_1 \cup Q_0$ and the initial states as $I = \{x, y\}$, where

$$\begin{aligned} Q_1 &= \{x, +0, +1, +2, +4, +8\}, \\ Q_0 &= \{y, -0, -1, -2, -4, -8\}. \end{aligned}$$

For every $b \in \{0, 1\}$ and every $q \in Q_b$, we define the output of q as $O(q) = b$.

Let us now describe the transitions of \mathcal{P} and explain how the protocol works. We give a (partial) graphical representation of \mathcal{P} as a Petri net in Figure 5. For readers not familiar with Petri nets, see Figure 6.

The agents, as a whole population, carry the binary representation of two numbers: a positive one (on the left of Figure 5) and a

negative one (on the right of Figure 5). Each agent initially in x (resp. y) is eventually converted into the binary representation of 5 (resp. -3). Two states of the form $\pm 2^i$, where $i \in \{0, 1, 2\}$, can be “promoted” into the equivalent representation made of states $\pm 2^{i+1}$ and ± 0 . Similarly, states of the form $\pm 2^{i+1}$ and ± 0 , where $i \in \{0, 1, 2\}$, can be “downgraded” into the equivalent representation made of two occurrences of state $\pm 2^i$. Assuming there are sufficiently many agents in states $+0$ and -0 , this allows for representing all possible representations of the two current numbers. The positive and negative numbers may “cancel out” with the middle transitions of Figure 5.

In any fair execution, one side will eventually “win” and all agents will end up in states $Q_1 \cup \{-0\}$ or $Q_0 \cup \{+0\}$ depending on whether $5x - 3y > 0$ holds or not. Extra transitions, not depicted in Figure 5, are added to the protocol to stabilize to a consensus. In more details, any state of $Q_1 \setminus \{+0\}$ can convert -0 into $+0$, and any state of Q_0 can convert $+0$ into -0 . Note that these rules are not entirely symmetric because of the case $5x - 3y = 0$.

By a careful analysis, it can be shown that if $4k + 2$ leaders are initially distributed equally among states $+0$ and -0 , then the protocol can never “run out of zeros”, regardless of the number of agents, and hence the protocol is correct.

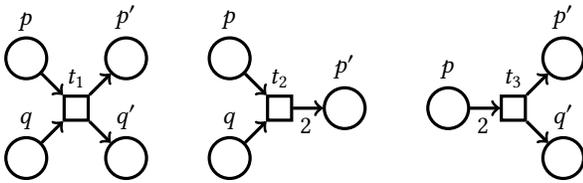


Figure 6. Left to right: Petri net representation of transitions $t_1: p, q \mapsto p', q'$, $t_2: p, q \mapsto p', p'$ and $t_3: p, p \mapsto p', q'$ respectively, where p, q, p', q' denote distinct states.

7 PEREGRINE: a tool for the analysis of population protocols

All the demos you have (hopefully) run while reading this paper execute code from PEREGRINE, a tool under development at the Chair for Foundations of Software Reliability of the Technical University of Munich. The current version of the tool, presented almost simultaneously with this paper at the CAV 2018 conference [9], is available at <https://peregrine.model.in.tum.de>.

PEREGRINE allows you to specify protocols either through an editor or as simple scripts, and to analyze them via a graphical interface. The scripts are particularly useful to describe families of protocols, like the ones discussed in Section 6 for the predicates $x \geq c$, or for linear inequalities. The analysis features of PEREGRINE include manual step-by-step or random simulation (the feature most used in this paper), automatic sampling (like the one used to analyze Sensei II’s majority protocol), statistics generation of average convergence speed, detection of incorrect executions through simulation, and formal verification of correctness. The formal verification component implements the verification algorithm for silent protocols presented in [10] and sketched in Sections 4.3 and 4.4. More precisely, the algorithm can automatically check whether a silent protocol computes a given predicate. If the answer is negative,

PEREGRINE returns a counterexample execution leading from an initial configuration to a terminal configuration which is either not in consensus, or is in the wrong consensus. The verification algorithm calls the SMT solver Z3 [15] to test satisfiability of Presburger formulas, and the LoLA model checker to find counterexamples [26].

We have used PEREGRINE to prove correctness of most of the protocols described in this paper. Actually, almost none of them was correct in our first attempt. Also, the typo in the average-and-conquer majority protocol of Figure 1 was found thanks to PEREGRINE, after it reported that the algorithm, when written as shown in the figure, was not correct. However, there is still much to do. At the time of writing these lines, PEREGRINE’s verification component only handles silent, leaderless protocols – while, for example, the succinct protocol for linear inequalities of Section 6.5 is neither – and the automatic time analysis of Section 5.3 is still a prototype. Looking ahead, the distributed computing community has defined a large number of extensions to the basic model, none of which has been yet studied by the verification community.

Population protocols are an excellent testbed for studying the parameterized verification (*i.e.*, verification for all possible population sizes) of liveness properties under fairness assumptions, and the quantitative verification of parameterized systems. The field is still in its infancy, and there is much work ahead before ninjas can be sure that, whatever they should agree on, they will agree before dawn.

Acknowledgments

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

Antonín Kučera³ was supported by the Czech Science Foundation, grant No. P202/12/G061.

We thank Dejvuth Suwimonteerabuth for implementing a first simulator of population protocols, and Philipp J. Meyer, Philip Offtermatt and Amrita Suresh for helpful discussions.

References

- [1] Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. 2017. Time-Space Trade-offs in Population Protocols. In *Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2560–2579. <https://doi.org/10.1137/1.9781611974782.169>
- [2] Dan Alistarh, Rati Gelashvili, and Milan Vojnović. 2015. Fast and Exact Majority in Population Protocols. In *Proc. ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 47–56. <https://doi.org/10.1145/2767386.2767429>
- [3] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. 2006. Computation in networks of passively mobile finite-state sensors. *Distributed Computing* 18, 4 (2006), 235–253. <https://doi.org/10.1007/s00446-005-0138-3>
- [4] Dana Angluin, James Aspnes, and David Eisenstat. 2006. Stably computable predicates are semilinear. In *Proc. 25th ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 292–299. <https://doi.org/10.1145/1146381.1146425>
- [5] Dana Angluin, James Aspnes, and David Eisenstat. 2008. Fast computation by population protocols with a leader. *Distributed Computing* 21, 3 (2008), 183–199. <https://doi.org/10.1007/s00446-008-0067-z>
- [6] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. 2007. The computational power of population protocols. *Distributed Computing* 20, 4 (2007), 279–304. <https://doi.org/10.1007/s00446-007-0040-2>
- [7] James Aspnes and Eric Ruppert. 2009. An Introduction to Population Protocols. In *Middleware for Network Eccentric and Mobile Applications*. Springer, Chapter 5, 97–120.
- [8] Michael Blondin, Javier Esparza, and Stefan Jaax. 2018. Large Flocks of Small Birds: on the Minimal Size of Population Protocols. In *Proc. 35th Symposium*

³The presented results were partially achieved during the author’s stay at TU München supported by the Friedrich Wilhelm Bessel Research Award (Alexander von Humboldt Foundation).

- on *Theoretical Aspects of Computer Science (STACS)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 16:1–16:14. <https://doi.org/10.4230/LIPIcs.STACS.2018.16>
- [9] Michael Blondin, Javier Esparza, and Stefan Jaax. 2018. Peregrine: A Tool for the Analysis of Population Protocols. In *Proc. 30th International Conference on Computer Aided Verification (CAV)*. Springer, to appear.
- [10] Michael Blondin, Javier Esparza, Stefan Jaax, and Philipp J. Meyer. 2017. Towards Efficient Verification of Population Protocols. In *Proc. 36th ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 423–430. <https://doi.org/10.1145/3087801.3087816>
- [11] Stephen P. Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. 2006. Randomized gossip algorithms. *IEEE Transactions on Information Theory* 52, 6 (2006), 2508–2530. <https://doi.org/10.1109/TIT.2006.874516>
- [12] Aaron R. Bradley and Zohar Manna. 2007. *The Calculus of Computation – Decision Procedures with Applications to Verification*. Springer. <https://doi.org/10.1007/978-3-540-74113-8>
- [13] Ioannis Chatzigiannakis, Othon Michail, and Paul G. Spirakis. 2010. Algorithmic Verification of Population Protocols. In *Proc. 12th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. Springer, 221–235. https://doi.org/10.1007/978-3-642-16023-3_19
- [14] Julien Clément, Carole Delporte-Gallet, Hugues Fauconnier, and Mihaela Sighireanu. 2011. Guidelines for the Verification of Population Protocols. In *Proc. International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society, 215–224. <https://doi.org/10.1109/ICDCS.2011.36>
- [15] Leonardo Mendonça de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Proc. 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24 Z3 is available at <https://github.com/Z3Prover/z3>.
- [16] Yuxin Deng and Jean-François Monin. 2009. Verifying Self-stabilizing Population Protocols with Coq. In *Proc. Third IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE)*. IEEE Computer Society, 201–208. <https://doi.org/10.1109/TASE.2009.9>
- [17] Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. 2017. Verification of population protocols. *Acta Informatica* 54, 2 (2017), 191–215. <https://doi.org/10.1007/s00236-016-0272-3>
- [18] Javier Esparza, Ruslán Ledesma-Garza, Rupak Majumdar, Philipp J. Meyer, and Filip Nikšić. 2014. An SMT-Based Approach to Coverability Analysis. In *Proc. 26th International Conference on Computer Aided Verification (CAV)*. Springer, 603–619. https://doi.org/10.1007/978-3-319-08867-9_40
- [19] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV)*. Springer, 585–591. https://doi.org/10.1007/978-3-642-22110-1_47 PRISM is available at <http://www.prismmodelchecker.org>.
- [20] Jérôme Leroux and Sylvain Schmitz. 2015. Demystifying Reachability in Vector Addition Systems. In *Proc. 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society, 56–67. <https://doi.org/10.1109/LICS.2015.16>
- [21] Ernst W. Mayr and Albert R. Meyer. 1982. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics* 46, 3 (1982), 305–329. [https://doi.org/10.1016/0001-8708\(82\)90048-2](https://doi.org/10.1016/0001-8708(82)90048-2)
- [22] Othon Michail and Paul G. Spirakis. 2018. Elements of the theory of dynamic networks. *Commun. ACM* 61, 2 (2018), 72. <https://doi.org/10.1145/3156693>
- [23] Philip Offtermatt. 2017. *A Tool for Verification and Simulation of Population Protocols*. Bachelor thesis. Technical University of Munich.
- [24] Jun Pang, Zhengqin Luo, and Yuxin Deng. 2008. On Automatic Verification of Self-Stabilizing Population Protocols. In *Proc. Second IEEE/IFIP International Symposium on Theoretical Aspects of Software Engineering (TASE)*. IEEE Computer Society, 185–192. <https://doi.org/10.1109/TASE.2008.8>
- [25] Mojżesz Presburger. 1929. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. *Comptes-rendus du premier congrès des mathématiciens des pays slaves* (1929), 192–201.
- [26] Karsten Schmidt. 2000. LoLA: A Low Level Analyser. In *Proc. 21st International Conference on Application and Theory of Petri Nets (ICATPN)*. Springer, 465–474. https://doi.org/10.1007/3-540-44988-4_27 LoLA is available at <http://service-technology.org/lola/>.
- [27] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. 2009. PAT: Towards Flexible Verification under Fairness. In *Proc. 21st International Conference on Computer Aided Verification (CAV)*. Springer, 709–714. https://doi.org/10.1007/978-3-642-02658-4_59