

# Population Protocols: Beyond Runtime Analysis<sup>\*</sup>

Javier Esparza<sup>1</sup>[0000-0001-9862-4919]

Technical University of Munich, Germany  
 esparza@in.tum.de

**Abstract.** We survey our recent work on the verification of population protocols and their state complexity.

## 1 Introduction

Population protocols are a model of computation in which an arbitrary number of indistinguishable finite-state agents interact in pairs to collectively decide if their initial global configuration satisfies a given property. Population protocols were introduced by Angluin *et al.* in [7,8] to study the theoretical properties of networks of mobile sensors with very limited computational resources, but they are also very strongly related to chemical reaction networks [53], a discrete model of chemistry in which agents are molecules that change their states due to collisions.

Population protocols decide a property by *stable consensus*. Each state of an agent is assigned a binary output (yes/no). At each step, a pair of agents is selected uniformly at random and allowed to interact. In a correct protocol, all agents eventually reach with probability 1 the set of states whose output correctly answers the question “did our initial configuration satisfy the property?” and stay in these states forever.

The *parallel runtime* of a protocol is defined as the expected number of interactions until a stable consensus is reached (i.e. until the property is decided), divided by the number of agents. In recent years, much research on population protocols has focused on the runtime of population protocols, and several landmark results have been obtained. In particular, recent results have studied protocols for majority and leader election in which the number of states grows with the number of agents, and shown that poly-logarithmic time is achievable by protocols without leaders, even for very slow growth functions, see e.g. [3,4,40]. Many of these results have been described in excellent surveys [5,27].

My work on population protocols, carried out with many of my PhD students and postdocs, and in collaboration with other colleagues, has focused on other aspects than runtime analysis, and this is the reason for the title of this note. I first

---

<sup>\*</sup> The work surveyed in this note was supported by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program under grant agreement No 787367 “Parameterized Verification and Synthesis” (PaVeS).

learned about population protocols at a talk by Paul Spirakis in ICALP 2008. As a researcher working on the theory of Petri nets, I noticed the connection of population protocols to Petri nets, and as a researcher working on automatic verification, I asked myself if the correctness problem—given a population protocol and a property, does the protocol decide the property?—was decidable. The problem consisted of checking whether an infinite collection of finite-state Markov chains, one for each initial configuration of the protocol, satisfies a rather sophisticated liveness property with probability 1. This makes it very hard: liveness properties are harder to verify than safety properties, probabilistic systems are harder than non-probabilistic ones, and parameterized problems, i.e., problems involving families of systems with an arbitrarily large number of agents, are *much* harder to verify than systems with a fixed number of agents. After looking at the problem for some time I could not find an answer, but I kept it at the back of my mind, and in 2015 (seven years later!) I suggested to my colleagues Pierre Ganty, Jérôme Leroux, and Rupak Majumdar to examine it again. This time, thanks to new progress by Leroux on the theory of Petri nets, we proved that the correctness problem is decidable, although as hard as the reachability problem for Petri nets [31]. This was the starting point of a research program devoted to the theory and practice of verifying population protocols, which reached an important milestone in 2020 with the release of PEREGRINE 2.0, a verifier based on new theoretical results [15,32]. The first part of this note surveys this research, adding all the work carried out since 2017 to a brief previous survey [28].

In 2018, Michael Blondin, Stefan Jaax and myself observed that a well-known result of the theory of Petri nets had a surprising application to the theory of population protocols: It showed that an infinite family of predicates of the form  $x \geq k$  for certain values of  $k$  could be decided by extremely succinct protocols with only  $O(\log \log k)$  states. This sparked our interest in the question of how many states are needed to decide a given predicate, or, by analogy to automata theory, the *state complexity* of the predicate. The question is relevant. For example, the fast protocol for majority implicitly described in [10] has tens of thousands of states. This is an obstacle to implementations of protocols in chemistry, where the number of states corresponds to the number of chemical species participating in the reactions. The number of states is also important because it plays the role of memory in sequential computational models. Indeed, the total memory available to a population protocol is the logarithm of the number of states multiplied by the number of agents.

To the best of our knowledge, we are the first group to study the state complexity of predicates. While we do not have a complete characterization yet, we have already proved several results. In 2018 the only bounds on state complexity were the ones derived from the synthesis procedures of [9,10]. The input to these procedures is a boolean combination of atomic predicates of the form  $\sum_{i=1}^k a_i x_i \sim b$ , where  $a_i$  and  $b$  are integers, and  $\sim$  is either  $<$  or  $\equiv_m$ , the latter denoting congruence modulo  $m$ . (It is known that every predicate decidable by population protocols is of this form.) The bounds of [9,10] are exponential in both the number of atomic predicates, and in their size, with numbers written

in binary. Since 2018 we have shown that every predicate has a protocol with a polynomial number of states both in the number and the size of the atomic predicates [16,14], and that, as mentioned above, some predicates have much smaller protocols [16]. Very recently, we have also obtained lower bounds for the state complexity [24]. The second part of the note surveys this work.

The note is structured as follows. Section 2 introduces terminology, and Sections 3 and 4 survey our work on verification and state complexity, respectively.

## 2 Some terminology

We assume that the reader is familiar with the basics of the population protocol model; here we just fix some terminology.

*Population protocols.* A population protocol has a set of states and transitions, with a distinguished set of initial states. Every state also has an associated output, 1 or 0. Transitions model interactions between two agents. They have the form  $q_1, q_2 \mapsto q'_1, q'_2$ , meaning that two agents in states  $q_1$  and  $q_2$  interact and move to states  $q'_1$  and  $q'_2$ . We assume that there is exactly one transition for each pair of states, but transitions can also be silent, meaning that the states of the agents do not change. A configuration of a protocol is a mapping assigning to each state a number of agents. Initial configurations put arbitrarily many agents in the initial states, and 0 agents elsewhere. A protocol starts at some initial configuration, and executes steps by repeatedly picking two agents uniformly at random and applying the corresponding transition. A run is an infinite sequence of configurations obtained by executing infinitely many steps.

A configuration has consensus 1 resp. 0 if all its agents occupy states with output 1 resp. 0. We also say that the configuration is a 1-consensus, resp. a 0-consensus, or just a consensus when we are not specific. A configuration is a *stable* 1-consensus if every configuration reachable from it, including itself, is a 1-consensus. A protocol is *well specified* if for every initial configuration  $C$  there is  $b \in \{0,1\}$  such that runs starting at  $C$  eventually reach a stable  $b$ -consensus with probability 1 (abbreviated as w.p.1 in this note); in this case, we say that the protocol *outputs*  $b$  for the initial configuration  $C$ . A well specified protocol with initial states  $q_1, \dots, q_k$  *decides* the predicate  $\varphi(x_1, \dots, x_k)$  defined by:  $\varphi(n_1, \dots, n_k) = b$  iff the protocol outputs  $b$  for the initial configuration that puts  $n_1, \dots, n_k$  agents in  $q_1, \dots, q_k$ .

We also consider protocols *with (one or more) leaders*. Intuitively, this is a population protocol with a set of distinguished agents. Formally, a protocol with leaders only differs from a normal population protocol in the definition of the set of initial configurations. The initial configurations of normal protocols put arbitrarily many agents in the initial states, and 0 agents elsewhere. In a protocol with leaders the initial configurations also put a fixed number of agents, the same for every initial configuration, in other states.

Population protocols can be seen as special classes of Petri nets or Vector Addition Systems [52,51]. For the purposes of this note, it suffices to say that,

like a protocol, a Petri net has a set of states (called *places* in Petri net jargon) and transitions. However, transitions have the form  $q_1, \dots, q_n \mapsto q'_1, \dots, q'_m$  for arbitrary  $n, m \geq 0$ . So, intuitively, transitions of a Petri net can represent interactions between more than two agents, and they can create or destroy agents. The theory of Petri nets has produced numerous results about the properties of their reachability graphs, i.e., the graphs with the configurations as nodes, and the steps as transitions. Such results can be immediately translated to population protocols.

*Presburger arithmetic, Presburger predicates, and Presburger sets.* A fundamental result of the theory of population protocols is that they decide precisely the Presburger predicates, i.e., the predicates expressible in Presburger arithmetic [11]. We briefly recall the definition of Presburger arithmetic and some results, and refer to [41] for more details.

Atomic formulas of Presburger arithmetic are of the form  $\sum_{i=1}^k a_i x_i \sim b$ , where  $a_i$  and  $b$  are integers,  $x_i$  are variables, and  $\sim$  is either  $<$  or  $\equiv_m$ , the latter denoting the congruence modulo  $m$  for some  $m \geq 2$ . We call these atomic formulas *threshold* and *remainder* (or *modulo*) formulas, respectively. The formulas of Presburger arithmetic are the result of closing atomic formulas under Boolean operations and first-order existential or universal quantification. For example,

$$\varphi(x, y) = \forall z \exists u: 3x - u \leq 0 \wedge 2z - y + u \geq 4 \wedge (x + y) \equiv_5 3$$

is a formula of Presburger arithmetic with free variables  $x$  and  $y$ .

A formula  $\varphi(x_1, \dots, x_n)$  is interpreted over  $\mathbb{N}^n$  in the expected way. A set of vectors  $S \subseteq \mathbb{N}^n$  is a *Presburger set* if there is a Presburger formula  $\varphi$  such that a vector belongs to  $S$  iff it satisfies  $\varphi$ , and a predicate over  $\mathbb{N}^n$  is a *Presburger predicate* if the set of vectors satisfying the predicate is a Presburger set. So Presburger formulas are finite representations of the Presburger sets and predicates. *Semilinear sets* are another representation of the Presburger sets, that is, a set is semilinear iff it is Presburger. In this note we do not need any specific properties of the semilinear representation.

Presburger arithmetic has a quantifier-elimination procedure, meaning that every formula can be transformed into an equivalent boolean combination of threshold and remainder predicates. The satisfiability problem for full Presburger arithmetic is decidable, but its complexity is high, it lies between 2-NEXP and 2-EXPSpace. For quantifier-free formulas the problem is NP-complete, and there exist SMT-solvers efficient in practice. The tools PEREGRINE 1.0 and 2.0 described later in this note are implemented on top of the Z3 solver [50].

### 3 Verification of population protocols

The design of population protocols is quite error prone. In our experience, it is hardly ever the case that the first design for a protocol computing a predicate is

correct. The problem is accentuated by the lack of a suitable high-level programming language for protocols, which makes their design akin to writing machine code.

In this context, the limited expressive power of population protocols also has a positive side: the correctness problem is not trivially undecidable, as happens with many other models of computation. In this section we show that the problem is in fact decidable, and survey our work leading from the first decidability result to the current decision procedures and to their practical implementation.

### 3.1 Decidability and complexity

In [31] we proved that the two central verification problems for population protocols are decidable:

- Well-specification: Given a population protocol, is it well specified?
- Correctness: Given a population protocol and a Presburger predicate (represented as a Presburger formula), does the protocol compute the predicate?

The results were extended to decidability of more general properties in [30]. The proofs proceed by reduction to the reachability problem for Petri nets, which is decidable [47,43]. We also showed that well-specification and correctness are recursively equivalent to the reachability problem for Petri nets. More precisely:

- The reachability problem for Petri nets can be reduced to well-specification or correctness in polynomial time;
- Given an oracle for the reachability problem for Petri nets, well-specification and correctness can be decided in elementary time, i.e., in time bounded by a finite tower of exponentials.

It has been shown recently that the reachability problem for Petri nets is Ackermann-complete, meaning that its complexity is bounded from below and from above by non-primitive recursive fast growing functions related to the Ackermann function [25,46,26]. Therefore, well-specification and correctness are not primitive recursive either, that is, no algorithm running in time bounded by a primitive recursive function can solve them.

The very high complexity of the correctness problem leads to the question whether the problem could be more tractable for special protocol classes. In [11] Angluin *et al.* not only characterized the expressive power of population protocols, but also of other models with more restricted communication mechanisms. In [36,33] we conducted a complete analysis of the complexity of the correctness problem for the models of [11]. We showed that for models based on immediate and delayed observation the correctness problem is PSPACE-complete and complete at the second level of the polynomial hierarchy, respectively. *Immediate observation* protocols have transitions of the form  $q_1, q \mapsto q_2, q$ . Intuitively, an agent in state  $q_1$  *observes* that the other process is in state  $q$ , which allows it to immediately move to state  $q_2$ . Intuitively, in such protocols if an agent at  $q_1$  can execute the transition, then every agent at  $q_1$  can take it as well, which greatly simplifies the verification task. In *delayed observation* protocols the observer in  $q_1$  may move to  $q_2$  at a later point.

### 3.2 A first attempt at a verification tool

In [18], published in 2017, we addressed the problem of developing an algorithm that would be efficient enough to automatically prove correctness for a class of protocols satisfying three conditions:

- (a) No loss of expressive power: the class should compute all Presburger predicates.
- (b) Naturality: the class should contain many of the protocols discussed in the literature.
- (c) Feasible membership problem: deciding if a protocol belongs to the class should have reasonable complexity.

In the paper we introduced the class of *Well-Specified Strongly Silent* protocols (WSSS). Intuitively, a protocol is silent if an execution reaches a terminal configuration with probability 1, where a configuration is terminal if cannot reach any other configuration. (Observe that a protocol correctly deciding a property need not be silent; indeed, the definition of when a protocol decides a property only requires that an execution reaches a *consensus* w.p.1. Reaching a consensus is a weaker property, because it allows the protocol to keep visiting different configurations, as long as in all of them the agents agree on the same value.) Further, a protocol is strongly silent if, loosely speaking, its transitions are organized in layers such that transitions of higher layers cannot be enabled by executing transitions of lower layers. In particular, if the protocol reaches a configuration of the highest layer that does not enable any transition, then this configuration is terminal. We showed that WSSS protocols satisfy (a) and (b), and proved that the membership problem for the class is in the complexity class DP. Recall that DP is the class of languages  $L$  such that  $L = L_1 \cap L_2$  for some languages  $L_1 \in \text{NP}$  and  $L_2 \in \text{coNP}$ ; in view of the Ackermannian complexity of the general case, this result shows that WSSS satisfies (c). The proof that the problem is in DP reduces membership to checking satisfiability resp. unsatisfiability of two quantifier-free formulas of Presburger arithmetic. The procedure was implemented in PEREGRINE 1.0 [17], a tool for the verification of population protocols built on top of the constraint solver Z3, and the first tool able to automatically prove well-specification for *all* initial configurations.

While WSSS protocols decide all Presburger predicates, PEREGRINE 1.0 had several limitations, which will be subject of the next section. From the most conceptual to the most practical:

- The verification algorithm of [18] was dissociated from the decidability results proved in [31]. To put it bluntly, the theoretical result was not guiding the design of a practical algorithm.
- The tool did not produce correctness certificates; if the tool returned “correct”, the user had to trust the result.
- Many protocols designed to be fast, or to use few states, are not in WSSS. Examples include the average-and-conquer protocol of [6] (for fixed values of the parameters), or the compact threshold protocols of [14]. In particular,

many protocols that perform a “random search”, like the second protocol in Example 1 below, are non-silent.

*Example 1.* The following two (very slow) protocols decide whether the number of blue agents minus the number of red agents is at least  $2^k$  for a given  $k \geq 1$ . These protocols are also of interest in the next section on state complexity.

*First protocol.* Each agent has a bag that can hold up to  $2^k$  pebbles and a flag that can be up or down (corresponding to output 1 and 0, respectively). Initially, each agent has one pebble and its flag is down. When two agents meet they update their bags and flags depending on their colors:

- Two red agents. No change.
- One blue and one red agent. If none of the two bags is empty, then both agents throw one pebble away and lower their flags; we call this interaction a *cancellation*. Else, if the bag of the blue agent is full (that is, if it has  $2^k$  pebbles) or if both flags were up before the interaction, then both agents raise their flags. Else, both agents lower their flags.
- Two blue agents. One agent gives the other as many pebbles as the other agent’s bag can still hold. If this fills the other agent’s bag, or if both flags were up before the interaction, then both agents raise their flags; otherwise they lower them.

Let us prove correctness. Let  $x$  and  $y$  be the numbers of blue and red agents, respectively. W.p.1 cancellations keep occurring until a configuration  $C$  is reached in which only blue or only red agents have pebbles (or no agent has pebbles). If  $x - y \geq 2^k$ , then no red agents have pebbles at  $C$ , and in runs from  $C$  some blue agent fills its bag and raises its flag w.p.1. The bag remains full forever, and w.p.1 this agent eventually meets all other agents *without any other interaction happening in-between*, after which all flags are up, and stay up forever. If  $x - y < 2^k$ , then after  $C$  no bag is ever full, and so any flag that is lowered stays down forever. Moreover, at  $C$  the flag of the blue agent that participated in the last cancellation is down, and this agent brings down the flag of any agent it meets. So eventually all flags stay down w.p.1.

*Second protocol.* Again, agents have bags and flags. The following updates ensure that the number of pebbles in each bag is always 0 or a power of 2:

- Two red agents. No change.
- One blue and one red agent. If both agents have *exactly one pebble*, they throw their pebbles away and lower their flags. Else, if the bag of the blue agent is full or both flags were up before the interaction, then both agents raise their flags. Else, both agents lower their flags.
- Two blue agents. If both agents hold *the same number of pebbles*, then one of them gives to the other as many pebbles as the other’s bag can still hold; if one agent has no pebbles, then it receives from the other half its pebbles; otherwise no pebbles are exchanged. If after this some bag is full, or if both flags were up before the interaction, then both agents raise their flags, else they lower them.



This protocol is also correct. Intuitively, blue agents can distribute pebbles among them into any combination of powers of 2 (up to  $2^k$ ). For example, if  $k = 2$  the blue agents can partition 5 pebbles among them as  $1+1+1+1+1$  (5 agents get one pebble each);  $2+1+1+1$ ;  $2+2+1$ ; or  $4+1$ . Randomness ensures that all these partitions are visited infinitely often, and so that cancellations keep occurring until a configuration  $C$  is reached in which only blue or only red agents have pebbles (or no agent has pebbles).

If  $x - y \geq 2^k$ , then runs from  $C$  eventually execute the following sequence of interactions w.p.1: first, some blue agent fills its bag and raises its flag; this agent then proceeds to meet all red agents, and then all blue agents whose bag is not empty; after that, the agent meets each blue agent with empty bag, sharing its pebbles with it, but only to recover them immediately. After this sequence all flags are up, and remain so forever. If  $x - y < 2^k$ , then the argument is as for the first protocol.

The first protocol is silent. If  $x \geq y$ , then w.p.1 it eventually reaches and stays in the configuration in which  $\lfloor (x - y)/2^k \rfloor$  blue agents have  $2^k$  pebbles, one blue agent has  $(x - y) \bmod 2^k$  pebbles each, all other agents have 0 pebbles, and all flags are up or down, depending on whether  $x - y \geq 2^k$  holds or not. If  $x \leq y$ , the protocol reaches and stays in the configuration in which  $y - x$  red agents have one pebble each, all other agents have 0 pebbles, and all flags are down. The second protocol needs exponentially fewer states, but is not silent. Indeed, when  $x \geq y$  the blue agents keep visiting all partitions of  $x - y$  forever.

### 3.3 A new proof methodology: stage graphs

Finding theoretical and algorithmic answers to the limitations of PEREGRINE 1.0 took three years. Initially we did not even have a clear picture of these limitations. They emerged when we started to investigate how to automatically compute an upper bound on the expected runtime of a protocol. This work, published in [19], introduced stage graphs, a notion that, after many rewrites, finally led to the stage graph proof methodology of [15], which we describe now.

Stage graphs reduce correctness to checking that certain assertions are inductive invariants, and that certain ranking functions decrease in appropriate ways. For standard sequential programs these checks are still undecidable problems, but for population protocols they turn out to be decidable.

**Certificates of correctness** The *reachability graph* of a population protocol has all possible configurations as nodes, and an edge from  $C$  to  $C'$  whenever  $C'$  is reachable from  $C$  in one step. It is an infinite graph, but every configuration has finitely many descendants. (A consequence of the fact that transitions do not change the number of agents.) We call graphs with this property *weakly finite* [29]. An edge of the graph corresponds to executing one transition of the protocol. The probability of executing a transition at configuration  $C$  is the fraction of the pairs of agents at  $C$  that enable the transition. Equipped with these notions, let us briefly review how to certify different kinds of properties.



*Certifying safety.* Safety properties can be certified using inductive invariants. An inductive invariant is just a set of configurations closed under the reachability relation, i.e., if a configuration belongs to the set, then so do all its successors. Given a set  $I$  of initial configurations and some set  $D$  of dangerous configurations, an inductive invariant  $Int$  satisfying  $I \subseteq Int$  and  $Int \cap D = \emptyset$  certifies that  $D$  is not reachable from  $I$ .

*Certifying termination.* Liveness properties, like termination, can be certified by ranking functions assigning to each configuration an element of a set with a well-founded order, like the natural numbers. Termination for *all* runs of the program starting at  $I$  is certified by an inductive invariant  $Int$  containing  $I$ , and a *strictly decreasing* ranking function over  $Int$ , i.e., a ranking function that strictly decreases whenever the protocol takes a step.

*Certifying termination w.p.1.* Termination *with probability 1* can also be certified by an inductive invariant  $Int$  containing  $I$ , and a ranking function  $f$ . However, the ranking function only needs to be *weakly decreasing*, meaning that for every non-terminal configuration  $C \in Int$ , some configuration  $C'$  reachable from  $C$  in one or more steps satisfies  $f(C') < f(C)$ . Indeed, if such a function exists, then for every non-terminal configuration of  $C \in Int$  there is a positive probability of taking a path that stays within  $Int$  by inductivity, and decreases the ranking function. Since the reachability graph is weakly finite, this probability is bounded from below by some  $\epsilon > 0$  independent of  $C$ . So runs reach and stay at terminal configurations w.p.1.

*Certifying stable consensus w.p.1.* In order to certify that a run starting at a given set  $I$  eventually reaches stable consensus  $b$  w.p.1, for some given  $b \in \{0, 1\}$ , we need two inductive invariants  $Int_1$ ,  $Int_2$  and a ranking function  $f$  satisfying three properties:

- $Int_1$  contains  $I$  (and so, by inductivity, also all configurations reachable from  $I$ ) and  $Int_2$ ;
- $Int_2$  contains only  $b$ -consensus configurations (and so, by inductivity, any run reaching  $Int_2$  reaches stable consensus); and
- $f$  is weakly decreasing on  $Int_1 \setminus Int_2$ , i.e., for every  $C \in Int_1 \setminus Int_2$ , some  $C'$  reachable from  $C$  in one or more steps satisfies  $f(C') < f(C)$ .

The same argument as above shows that runs starting at  $C \in Int_1 \setminus Int_2$  eventually reach  $Int_2$  w.p.1, and, since  $Int_2$  is inductive, get trapped in  $Int_2$ . Since  $Int_2$  only contains configurations with consensus  $b$ , runs starting at  $I$  reach stable consensus  $b$  w.p.1.

This proof technique is *complete*, meaning that if a run starting at  $I$  eventually reaches stable consensus  $b$  w.p.1, then we can always find a suitable  $Int_1$  and  $Int_2$ , and  $f$ . Indeed, it suffices to choose  $Int_1$  as the set of all configurations reachable from  $I$ ;  $Int_2$  as the set of all configurations of  $Int_1$  with stable consensus  $b$ ; and  $f$  as the function assigning 0 to the configurations of  $Int_2$ , and 1 to the configurations of  $Int_1 \setminus Int_2$ .

*Stage graphs.* It is useful to split stable consensus proofs into a small steps. For this one can exhibit a finite, directed, and acyclic graph, whose nodes are pairs  $v = (Int, f)$ , where  $Int$  is an inductive invariant, and  $f$  is a ranking function certifying that runs starting at  $Int$  eventually get trapped in  $Int_1 \cup \dots \cup Int_n$  w.p.1, where  $Int_1, \dots, Int_n$  are the invariants of the children of  $v$ . Further, the invariants of the bottom nodes of the graph only contain consensus configurations. In [15] we call these objects *stage graphs*, and their nodes *stages*. A stage graph proves that runs starting at a stage “travel down the graph w.p.1” until they reach a bottom stage, and so stable consensus. Intuitively, stages correspond to “milestones” towards stable consensus.

To prove a protocol correct, we produce two stage graphs proving that runs starting at initial configurations that satisfy the predicate eventually reach stable consensus 1, and the corresponding property for stable consensus 0. The stage graphs have an initial stage containing the initial configurations satisfying or not satisfying the predicate, respectively. Observe that, since stages are inductive the initial stages also contain every reachable configuration. Let us examine stage graphs in more detail with the help of a well-known example.

*Example 2.* Consider the following majority protocol, whose purpose is to decide if the initial configuration contains more blue agents than red agents. Apart from red or blue, agents can also be active or passive, yielding four possible states  $Q = \{\mathbf{B}, \mathbf{R}, \mathbf{b}, \mathbf{r}\}$  (uppercase for active agents, lowercase for passive ones). The initial states are  $\mathbf{B}$  and  $\mathbf{R}$ , and so initially all agents are active. The protocol has four transitions:

$$\begin{array}{ll} t_1: \mathbf{B}, \mathbf{R} \mapsto \mathbf{b}, \mathbf{r} & t_2: \mathbf{B}, \mathbf{r} \mapsto \mathbf{B}, \mathbf{b} \\ t_3: \mathbf{R}, \mathbf{b} \mapsto \mathbf{R}, \mathbf{r} & t_4: \mathbf{b}, \mathbf{r} \mapsto \mathbf{b}, \mathbf{b} \end{array}$$

The blue states  $\mathbf{B}, \mathbf{b}$  have output 1, and the red states  $\mathbf{R}, \mathbf{r}$  output 0. So in this case, for better visualization, we call the outputs “blue” and “red”, instead of 1 and 0. The protocol is correct if it satisfies the following property: for every initial configuration  $C$ , i.e., every configuration  $C$  satisfying  $C(\mathbf{b}) = C(\mathbf{r}) = 0$ , if  $C(\mathbf{B}) < C(\mathbf{R})$ , eventually all agents stay forever in the red states  $\{\mathbf{R}, \mathbf{r}\}$  w.p.1, and if  $C(\mathbf{B}) \geq C(\mathbf{R})$  eventually all agents stay forever in the blue states  $\{\mathbf{B}, \mathbf{b}\}$  w.p.1.

Figure 1 shows two stage graphs for the protocol. Stages are given as constraints over variables  $\mathbf{B}, \mathbf{R}, \mathbf{b}, \mathbf{r}$ , describing the number of agents in states  $\mathbf{B}, \mathbf{R}, \mathbf{b}, \mathbf{r}$ , respectively. For example, the constraint  $\mathbf{B} < \mathbf{R}$  represents the set of all configurations  $C$  satisfying  $C(\mathbf{B}) < C(\mathbf{R})$ . Ranking functions are described as functions of  $\mathbf{B}, \mathbf{R}, \mathbf{b}, \mathbf{r}$ . For example, the function  $\mathbf{B} + \mathbf{R}$  assigns to every configuration  $C$  the number  $C(\mathbf{B}) + C(\mathbf{R})$ .

The stage graph on the left of Figure 1 proves that runs starting at any configuration satisfying  $\mathbf{B} < \mathbf{R}$  reaches stable consensus red with probability 1. The “human” proof goes as follows: because of transition  $t_1$ , from any configuration satisfying  $\mathbf{B} < \mathbf{R}$  the protocol eventually reaches a configuration satisfying  $\mathbf{B} = 0$ , and then transition  $t_3$  eventually changes all remaining blue agents into red agents. The stage graph reflects this proof structure:

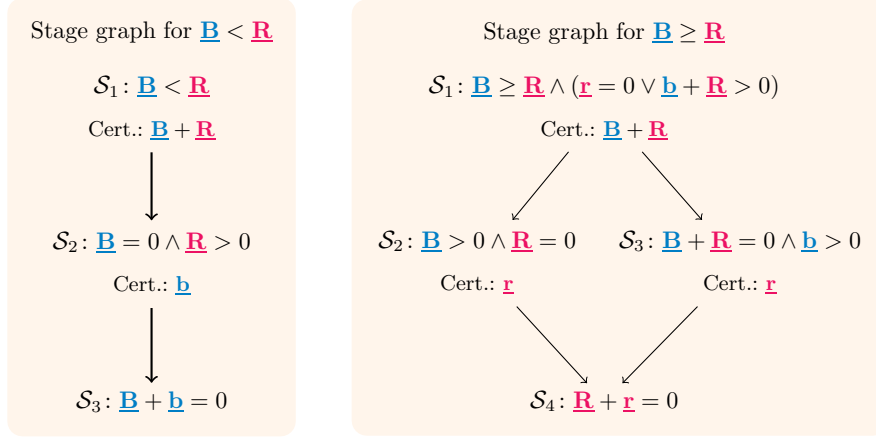


Fig. 1: Stage graphs for the protocol of Example 2

- The initial stage  $\mathcal{S}_1$  contains exactly the configurations satisfying  $\underline{B} < \underline{R}$ . The ranking function  $f(\underline{B}, \underline{R}, \underline{b}, \underline{r}) = \underline{B} + \underline{R}$  certifies that runs starting at a configuration of  $\mathcal{S}_1$  eventually get trapped in  $\mathcal{S}_2$ , the set of configurations satisfying  $\underline{R} > 0 \wedge \underline{B} = 0$ , w.p.1. Indeed, consider any configuration  $C \in \mathcal{S}_1 \setminus \mathcal{S}_2$ , i.e., a configuration satisfying  $0 < \underline{B} < \underline{R}$ . Then  $C$  enables transition  $t_1$ . Letting  $C \xrightarrow{t_1} C'$  we have

$$f(C') = \underline{B}' + \underline{R}' = (\underline{B} - 1) + (\underline{R} - 1) < \underline{B} + \underline{R} = f(C).$$

So  $f$  is weakly decreasing.

- Similarly, the ranking function  $g(\underline{B}, \underline{R}, \underline{b}, \underline{r}) = \underline{b}$  certifies that runs starting at a configuration of  $\mathcal{S}_2 \setminus \mathcal{S}_3$  eventually get trapped in  $\mathcal{S}_3$ . Since  $\mathcal{S}_3$  is the set of configurations without blue agents, we are done. Observe that not every transition decreases  $\underline{b}$ ; actually, transition  $t_4$ , which is enabled at some configurations of  $\mathcal{S}_2 \setminus \mathcal{S}_3$ , increases it. However,  $g$  is weakly decreasing because of transition  $t_3$ .
- Observe that  $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$  are inductive invariants. For example, if a configuration satisfies  $\underline{B} < \underline{R}$ , then so does any configuration reached by applying any of the four transitions of the protocol.

Let us now consider the stage graph on the right. It proves that runs starting at initial configurations satisfying  $\underline{B} \geq \underline{R}$  (that is, at the set of configurations satisfying  $\underline{B} \geq \underline{R}$  and  $\underline{b} = 0 = \underline{r}$ ) reach stable consensus blue w.p.1. The choice of the initial stage  $\mathcal{S}_1$  is not completely trivial.  $\mathcal{S}_1$  must satisfy three conditions: (a) contain all configurations satisfying  $\underline{B} \geq \underline{R}$  and  $\underline{b} = 0 = \underline{r}$ ; (b) contain only configurations with a majority of blue agents or a tie, because only those configurations reach stable consensus blue; (c) be inductive. We cannot choose  $\mathcal{S}_1 = \underline{B} \geq \underline{R}$  because it violates (b); for example, the configuration given by

$\underline{\mathbf{B}} = 1$ ,  $\underline{\mathbf{R}} = 1$ ,  $\underline{\mathbf{b}} = 1$ ,  $\underline{\mathbf{r}} = 2$  has a majority of red agents. We cannot choose  $\mathcal{S}_1 = \underline{\mathbf{B}} \geq \underline{\mathbf{R}} \wedge \underline{\mathbf{b}} = 0 = \underline{\mathbf{r}}$  either, because it violates (c). One has to find the right set between these two.

The “human” proof uses as milestone the configurations at which there are no agents left in state  $\mathbf{R}$ . These configurations can be of three kinds, corresponding to the stages  $\mathcal{S}_2$ ,  $\mathcal{S}_3$ , and  $\mathcal{S}_4$ :

- if there are no red agents left (stage  $\mathcal{S}_4$ ), then the run has already reached stable consensus blue;
- if there are agents left in  $\mathbf{B}$  (stage  $\mathcal{S}_2$ ), then any agents in state  $\mathbf{r}$  are eventually turned blue by transition  $t_2$ ;
- if there are agents left in  $\mathbf{b}$  (stage  $\mathcal{S}_3$ ), then any agents in state  $\mathbf{r}$  are eventually turned blue by transition  $t_4$ .

**Decidability of correctness.** If we are given stage graphs and told they prove that a protocol correctly decides a given property, we can in principle check this statement. We need to check that the initial stages contain the initial configurations satisfying and violating the property, respectively; that all stages are inductive and all ranking functions weakly decreasing; and that the bottom stages only contain configurations with the right consensus. However, since stages are infinite sets, the problem of carrying out such checks might be undecidable. The main theorem of [15] proves that, if a protocol is correct, then there exist stage graphs for which the checks reduce to proving satisfiability of formulas of Presburger arithmetic, which is decidable. More precisely, the theorem proves that every correct protocol has *Presburger stage graphs*, i.e., stage graphs satisfying the following properties:

- Stages are Presburger sets of configurations, i.e., sets expressible in Presburger arithmetic.
- Ranking functions are *Presburger functions*.  
A ranking function  $f$  is *Presburger* if there is a formula  $\varphi(\mathbf{C}, \mathbf{n})$  of Presburger arithmetic with free variables  $\mathbf{C}$  and  $\mathbf{n}$  such that for every configuration  $C$  and every number  $n$  we have  $f(C) = n$  iff  $\varphi(C, n)$  holds.
- Each ranking function  $f$  comes equipped with a bound  $B$  such that for every configuration  $C$  in the domain of  $f$ , some configuration  $C'$  reachable from  $C$  in at most  $B$  steps satisfies  $f(C') < f(C)$ . So, strictly speaking, a Presburger stage graph consists not only of stages and ranking functions, but also of bounds for these functions.

Let us now see why checking that a stage graph is Presburger reduces to the satisfiability problem of Presburger arithmetic. Checking that the initial stages contain all initial configurations, and that the bottom stages only contain configurations with the right consensus is easy, because the sets of initial configurations and consensuses are Presburger. Let us consider the other two checks.

*Checking that a stage is inductive.* Since stages are Presburger sets, given a stage  $S$  there is a formula  $S(\mathbf{C})$  expressing it. Further, for every transition  $t$  it is easy to construct a formula  $step_t(\mathbf{C}, \mathbf{C}')$  that holds iff  $C \xrightarrow{t} C'$ . For example, for our majority protocol we have

$$step_{t_1}(\mathbf{C}, \mathbf{C}') := \mathbf{C} \geq (1, 1, 0, 0) \wedge \mathbf{C}' = \mathbf{C} + (-1, -1, 1, 1)$$

So inductivity is expressed by the formula

$$\forall \mathbf{C}, \mathbf{C}': \left( S(\mathbf{C}) \wedge \bigvee_{t \in T} step_t(\mathbf{C}, \mathbf{C}') \right) \rightarrow S(\mathbf{C}')$$

where  $T$  is the set of transitions of the protocol.

*Checking that a ranking function is weakly decreasing.* If the reachability relation of population protocols would be expressible in Presburger arithmetic, i.e., if there were a Presburger formula  $reach(\mathbf{C}, \mathbf{C}')$  such that  $reach(C, C')$  holds iff  $C \xrightarrow{*} C'$ , then the weakly-decreasing property for arbitrary ranking functions would be expressible by the formula

$$\exists \mathbf{C}', \mathbf{n}, \mathbf{n}': reach(\mathbf{C}, \mathbf{C}') \wedge \varphi_f(\mathbf{C}, \mathbf{n}) \wedge \varphi_f(\mathbf{C}', \mathbf{n}') \wedge \mathbf{n}' \geq \mathbf{n}$$

However, this is not the case; it is well known that the reachability relation of Petri nets may not be Presburger, and the result easily transfers to population protocols. This is the reason for the restriction to bounded ranking functions. It is easy to construct by induction a formula  $reach_B(\mathbf{C}, \mathbf{C}')$  that holds if there exists a configuration  $C''$  reachable from  $C$  in at most  $B$  steps. Just take

$$\begin{aligned} reach_1(\mathbf{C}, \mathbf{C}') &:= \mathbf{C} = \mathbf{C}' \vee \bigvee_{t \in T} step_t(\mathbf{C}, \mathbf{C}') \\ reach_{k+1}(\mathbf{C}, \mathbf{C}') &:= \exists \mathbf{C}'': reach_1(\mathbf{C}, \mathbf{C}'') \wedge reach_k(\mathbf{C}'', \mathbf{C}') \end{aligned}$$

Now we can express the weakly decreasing property as above, replacing  $reach$  by  $reach_B$ .

As we mentioned before, the proof of existence of Presburger stage graphs is based on deep results on the theory of Petri nets, which can also be applied to population protocols. The main one is Leroux's theorem [45], stating the following. Let  $X$  and  $Y$  be Presburger sets of configurations of a Petri net, and let  $reach(X)$  be the set of configurations reachable from  $X$ . The theorem states that if  $reach(X) \cap Y = \emptyset$  holds, then there exists a Presburger inductive invariant that certifies this fact, i.e., there exists a Presburger set  $S$  closed under the reachability relation such that  $reach(X) \subseteq S$  and  $S \cap Y = \emptyset$ . Observe that if  $reach(X)$  were always itself a Presburger set, then we could just take  $S = reach(X)$ . Intuitively, Leroux's theorem shows that, while  $reach(X)$  is not always a Presburger set, it is always very close to it (in fact, Leroux's proof shows that  $reach(X)$  always belongs to a class of sets called *almost semilinear*).

### 3.4 Population protocols decide exactly the Presburger predicates

Angluin *et al.* proved in [11] that population protocols compute exactly the Presburger predicates. The difficult part is to show that population protocols *only* compute Presburger predicates. Let us show that this is a simple corollary of the fact that correctness can always be certified by Presburger stage graphs.

Consider a protocol that decides a predicate, say  $\varphi$ . Let  $I$  be the set of initial configurations of the protocol, and let  $I_1$  and  $I_0$  be a partition of  $I$  into the set of initial configurations that satisfy and do not satisfy  $\varphi$ , respectively. Our theorem shows that there exists a Presburger stage graph with initial stage  $\mathcal{S}$  such that  $I_1 \subseteq \mathcal{S}$ . This stage graph proves that a run starting at any configuration of  $\mathcal{S}$  eventually reaches stable consensus 1 w.p.1. Since the protocol decides  $\varphi$ , no configuration of  $I_0$  belongs to  $\mathcal{S}$ , i.e., we have  $\mathcal{S} \cap I_0 = \emptyset$ . Together with  $I_1 \subseteq \mathcal{S}$ , we have  $\mathcal{S} \cap I = I_1$ . But  $\mathcal{S}$  is Presburger, and so is  $I$  (indeed,  $I$  is just the set of configurations with 0 agents in non-initial states). Since Presburger sets are closed under intersection,  $I_1$  is also Presburger.

### 3.5 Automatic computation of stage graphs

We have developed a practical approach to the computation of stage graphs, implemented in the tool PEREGRINE 2.0 [32]. The design of the tool is guided by the theoretical results on stage graphs, and by the notion of *dead transitions*. A transition  $t$  is *dead at a configuration*  $C$  if no run starting at  $C$  executes  $t$ , and  $t$  is *dead at a stage*  $\mathcal{S}$  if  $t$  is dead at every  $C \in \mathcal{S}$ . Population protocols designed by humans usually run in phases. Initially, all transitions are alive, and the end of each phase is marked by the “death” of one or more transitions, i.e., by reaching a configuration at which these transitions become dead. Runs of the protocol keep “killing transitions” until they reach a consensus configuration whose consensus cannot be broken by any of the transitions still alive. This consensus is then stable. When applied to the majority protocol, PEREGRINE 2.0 computes automatically two stage graphs very similar to those of Figure 1 in a couple of seconds.

Like PEREGRINE 1.0, PEREGRINE 2.0 is built on top of the Z3 constraint solver. More precisely, it uses Z3 to check satisfiability of formulas of the existential fragment of Presburger arithmetic. The existential fragment is as expressive as full Presburger arithmetic, but can be handled much more efficiently.

Given a protocol and a Presburger predicate, PEREGRINE 2.0 computes two stage graphs, proving that runs starting at every initial configuration of the protocol satisfying (resp. violating) the predicate eventually reach stable consensus 1 (resp. stable consensus 0) w.p.1. Let  $I_1$  be the set of initial configurations satisfying the predicate, the other case being similar. PEREGRINE 2.0 maintains a worklist of Presburger stages, finitely represented by Presburger formulas. Initially, the worklist contains only one stage, namely an inductive Presburger overapproximation  $PotReach(I_1)$  (for “potentially reachable”) of the configurations reachable from  $I_1$ . The procedure computing  $PotReach(I_1)$  is the result of

many years of research on tractable “relaxations” of the reachability relation of Petri nets [35,34,39,20,13].

In its main loop, PEREGRINE 2.0 repeatedly picks a Presburger stage  $\mathcal{S}$  from the worklist, and processes it. First, the tool checks whether  $\mathcal{S}$  is terminal, i.e., if all its configurations are a 1-consensus. (Since  $\mathcal{S}$  is inductive by construction, an affirmative answer implies that all configurations of  $\mathcal{S}$  are stable 1-consensuses.) Checking that every configuration of  $\mathcal{S}$  is a 1-consensus reduces to checking unsatisfiability of a simple formula. If  $\mathcal{S}$  is not terminal, the tool attempts to construct one or more successor stages with strictly more dead transitions than  $\mathcal{S}$ . For this, the tool computes a set of *eventually dead transitions*: transitions that are alive at one or more configurations of  $\mathcal{S}$ , but will become dead w.p.1 in any run starting at those configurations. Again, the procedure to compute  $U$  makes heavy use of results of Petri net theory [42], but also of the theory of well-quasi-orders [1,38].

If PEREGRINE 2.0 finds a nonempty set of eventually dead transitions, then it constructs a successor stage of  $\mathcal{S}$  by overapproximating the configurations reachable from  $\mathcal{S}$ , underapproximating the configurations at which the transitions of  $U$  are dead, and intersecting the results. If PEREGRINE 2.0 fails to find eventually dead transitions, it heuristically splits  $\mathcal{S}$  into different stages and adds them to the worklist to be processed. Indeed, it could be the case that no transition becomes eventually dead from *every* configuration of  $\mathcal{S}$ , but this no longer holds after a split; for example, imagine that transition  $t_1$  eventually becomes dead from every configuration of  $\mathcal{S}_1 \subset \mathcal{S}$ , and another transition  $t_2$  becomes eventually dead from every configuration of  $\mathcal{S}_2 = \mathcal{S} \setminus \mathcal{S}_1$ . In this case, after splitting  $\mathcal{S}$  into  $\mathcal{S}_1$  and  $\mathcal{S}_2$  the tool can find nonempty sets of eventually dead transitions for both  $\mathcal{S}_1$  and  $\mathcal{S}_2$ .

PEREGRINE 2.0 has successfully proved correct a large variety of protocols, including majority and approximate majority protocols (Example 2, [19, Ex. 3], [6], [21], [44, *coin game*], [49]), various *flock-of-birds* protocol families ([22], [16, Sect. 3], [23, *threshold-n*]) for the family of predicates  $x \geq k$  for some constant  $k \geq 0$ ; or protocols for threshold and remainder predicates of [7,16]. For all these examples PEREGRINE 2.0 computes stage graphs with a few stages. Currently, the main limitation of the tool is the size of the systems of linear constraints involved, which limits the tool to protocols with up to some dozens of states and some thousands of transitions.

## 4 Succinct predicates and state complexity

After writing our first papers on the verification of population protocols, we observed that the theory of Petri nets was also relevant for a problem that, perhaps surprisingly, had not been studied yet: the *state complexity* of predicates decidable by population protocols. Informally, the state complexity of a predicate is the minimal number of states of the protocols that decide it, and, given a number  $\eta$ , one defines the function  $STATE(\eta)$  as the maximum state complexity of all predicates of size at most  $\eta$ . But what is the size of a predicate? This re-



quires us to fix a representation. Since population protocols compute exactly the predicates expressible in Presburger arithmetic [11], we must choose a representation of the Presburger sets. There are three natural representations: formulas of Presburger arithmetic, quantifier-free formulas of Presburger arithmetic, and semilinear sets [41]. Semilinear sets are difficult to parse by humans, and no paper on population protocols uses them to describe predicates. Full Presburger arithmetic is very succinct, but the complexity of its satisfiability problem lies between 2-NEXP and 2-EXPSpace [41], and so it can lead to results in which a predicate requires very few states, but only because of a representation that is very difficult to compute. This leaves quantifier-free Presburger arithmetic. This representation also has two advantages of its own. First, standard predicates for which numerous protocols have been given in the literature (like majority, threshold, or remainder predicates) are naturally expressed without quantifiers. Second, the procedures given so far to construct population protocols for any given Presburger predicate explicitly use the fact that Presburger arithmetic has a quantifier elimination procedure, i.e., they first construct protocols for all threshold and remainder predicates, and then show that the predicates computed by protocols are closed under negation and conjunction.

#### 4.1 State complexity: upper bounds

The first synthesis procedure for the construction of a protocol deciding a given Presburger predicate was presented in [9]. The procedure is simple and elegant, but it yields large protocols. Given a quantifier-free Presburger formula  $\varphi$ , i.e., a boolean combination of atomic formulas, the number of states of the synthesized protocol grows exponentially in both the number of bits of the largest coefficient of  $\varphi$  in absolute value, and the number of atomic formulas. In terms of  $|\varphi|$  (defined as the number of bits needed to write  $\varphi$ , with coefficients written in binary) they have  $\Omega(2^{\text{poly}(|\varphi|)})$  states. This raises the question whether protocols with  $O(\text{poly}(|\varphi|))$  states, which we call *succinct*, exist. We gave an affirmative answer in [14], completing first partial results obtained in [16]. We describe how to avoid both exponential dependencies.

*Handling large coefficients.* In order to prevent having the exponential dependence on the coefficients, we design protocols for threshold and remainder predicates that, loosely speaking, represent numbers in binary. A very easy case is described in Example 1: the first predicate for  $x - y \geq 2^k$  has  $\Theta(2^k)$  states, because agents can hold any number of pebbles between 0 and  $2^k$ , but the second has only  $\Theta(k)$  states, because the number of pebbles is always a power of 2. The construction of [14] proceeds in two steps: first we construct a succinct protocol in which the agents are assisted by *helpers*, additional agents that are not part of the input, and initially occupy a distinguished state, say  $H$ ; then we give an equivalent protocol without helpers. Helpers are similar to leaders, but with the property (guaranteed by the design of the protocol) that if the protocol works correctly with a certain number of helpers, then it also works correctly for any larger number. This property is crucial when dealing with boolean combinations.

Consider a predicate like  $13x - 9y \geq 5$ . Protocols for this predicate have two initial states, for  $x$  and  $y$ . Since  $13 = 2^3 + 2^2 + 2^0$ , the protocol has a transition that moves an agent in the initial state for  $x$  and two helpers into states  $2^3$ ,  $2^2$ , and  $2^0$ , respectively. (Transitions that move multiple agents to new states in one single step can be easily simulated by the usual protocol transitions with a very low cost in terms of states.) Similarly, another transition for each agent in the initial state for  $y$ , the protocol moves this agent and one helper into states  $-2^3$  and  $-2^1$ . Pairs of agents in states  $2^l$  and  $-2^l$  can “cancel”, meaning that they both move to state  $H$ , and so become helpers ready to continue assisting.

In [14] we show that this idea can be applied to every threshold or remainder predicate, resulting in a succinct protocol with a *fixed* number of helpers, cubic in the size of the predicate, but independent of the size of the input. But how do we go from this protocol to another one without helpers? For large inputs in which the number of agents exceeds this number of helpers, we can let each agent take two jobs: act as a regular agent *and* a helper. Let us show how to do for  $h$  helpers. In a first phase, the protocol assigns to each agent a number between 1 and  $h$ , ensuring that each number is assigned to at least one agent (this is the point at which we need a sufficiently large input with at least  $h$  agents). More precisely, at the end of this phase each agent is in a state of the form  $(x, i)$ , meaning that the agent initially represented one unit of input for variable  $x$ , and that it has been assigned number  $i$ . For this, initially every agent is placed in state  $(x, 1)$ . Transitions of the form  $(x, i), (x, i) \mapsto (x, i + 1), (x, i)$  for every  $1 \leq i \leq h - 1$  guarantee that all but one agent is promoted to  $(x, 2)$ , all but one to  $(x, 3)$ , etc. In other words, at each step one agent is “left behind”, and so the protocol has at least  $h$  helpers.

However, the protocol must be correct for all inputs, not only for those with at least  $h$  agents. In [14] this is solved by designing a second family of protocols for small inputs, which works in a completely different way. It is then easy to combine the protocols for large and small inputs into a protocol for all inputs.

*Handling large boolean combinations of atomic formulas.* The second problem of the synthesis procedure of [9] is the exponential dependence of the number of states on the number of atomic formulas. The dependence comes from the fact that, given protocols  $\mathcal{P}_1, \dots, \mathcal{P}_k$  with  $n_1, \dots, n_k$  states deciding formulas  $\varphi_1, \dots, \varphi_k$ , respectively, the synthesis procedure yields a protocol  $\mathcal{P}$  for deciding  $\varphi_1 \wedge \dots \wedge \varphi_k$  with  $n_1 \cdot n_2 \cdot \dots \cdot n_k$  states (and similarly for  $\varphi_1 \vee \dots \vee \varphi_k$ ). Intuitively, in  $\mathcal{P}$  each agent carries out  $k$  jobs: act as an agent of  $\mathcal{P}_1$ , of  $\mathcal{P}_2$ , ..., and of  $\mathcal{P}_k$ . The state of an agent is a  $k$ -tuple of states of the  $\mathcal{P}_1, \dots, \mathcal{P}_k$ , and when two agents meet, they compare their states in each protocol  $\mathcal{P}_i$ , and apply the corresponding transition. In other words: the new protocol executes all of  $\mathcal{P}_1, \dots, \mathcal{P}_k$  synchronously.

We need a new succinct construction for a boolean combination of atomic predicates with  $O(n_1 + \dots + n_k)$  instead of  $\Omega(n_1 \cdot \dots \cdot n_k)$  states. A naive first idea is to let  $\mathcal{P}$  execute  $\mathcal{P}_1, \dots, \mathcal{P}_k$  asynchronously in parallel, instead of synchronously, and combine the results. However, this does not work. Assume  $\varphi_1, \dots, \varphi_k$  have arity  $m$ . In order to compute  $(\varphi_1 \wedge \dots \wedge \varphi_k)(\mathbf{x})$ , where  $\mathbf{x} = (x_1, \dots, x_m)$ , the

protocol  $\mathcal{P}$  would have to dispatch  $\mathbf{x}$  agents to (the input states of) each  $\mathcal{P}_i$ , giving a total of  $k \cdot \mathbf{x}$  agents, but the protocol only has the agents of the input, i.e.  $\mathbf{x}$  agents. But couldn't we use  $(k-1) \cdot \mathbf{x}$  helpers, and then obtain an equivalent protocol without helpers? No, this does not work either, because in this case the number of helpers depends on the size of the input, and the technique we used above only allows us to simulate a fixed number of helpers. The solution is to use a more sophisticated construction for parallel asynchronous computation. Again, we need to consider separately the cases of large and small inputs, but the former is more interesting, and so we only describe protocols for large inputs.

Given an arbitrary threshold or modulo predicate  $\psi(\mathbf{x})$  of arity  $m$ , it is easy to construct a predicate  $\tilde{\psi}(\mathbf{y}, \mathbf{z})$  of arity  $2m$  satisfying

$$\tilde{\psi}(\mathbf{y}, \mathbf{z}) = \psi(k\mathbf{y} + \mathbf{z})$$

For instance, if  $\psi(x_1, x_2) = (3x_1 - 2x_2 > 6)$  and  $k = 4$ , then we can choose  $\tilde{\psi}(y_1, y_2, z_1, z_2) = (12y_1 + 3z_1 - 8y_2 - 2z_2 > 6)$ .

Intuitively, the idea is to let  $\mathcal{P}$  compute  $\tilde{\varphi}_1(\mathbf{y}, \mathbf{z}), \dots, \tilde{\varphi}_k(\mathbf{y}, \mathbf{z})$  instead of  $\varphi_1(\mathbf{x}), \dots, \varphi_k(\mathbf{x})$  for some  $\mathbf{y}$  and  $\mathbf{z}$  satisfying  $\mathbf{x} = k\mathbf{y} + \mathbf{z}$ . Then  $\mathcal{P}$  only needs to dispatch a total of

$$k \left( \sum_{i=1}^m y_i + z_i \right) = k \left( \sum_{i=1}^m y_i + (x_i - ky_i) \right) \leq \sum_{i=1}^m x_i + m \cdot (k-1)^2$$

agents to compute all of  $\tilde{\varphi}_1, \dots, \tilde{\varphi}_k$ . So  $\mathcal{P}$  only needs  $m \cdot (k-1)^2$  helpers, a fixed number independent of the number of agents.

Let us now describe how  $\mathcal{P}$  computes  $\tilde{\varphi}_i(\mathbf{y}, \mathbf{z})$  for some  $\mathbf{y}$  and  $\mathbf{z}$  satisfying  $\mathbf{x} = k\mathbf{y} + \mathbf{z}$ . Let  $\tilde{\mathcal{P}}_1, \dots, \tilde{\mathcal{P}}_k$  be protocols computing  $\tilde{\varphi}_1, \dots, \tilde{\varphi}_k$ , let  $\mathbf{x}_1, \dots, \mathbf{x}_m$  be the input states of  $\mathcal{P}$ , and let  $\mathbf{y}_1^j, \dots, \mathbf{y}_m^j$  and  $\mathbf{z}_1^j, \dots, \mathbf{z}_m^j$  be the input states of  $\tilde{\mathcal{P}}_j$  for every  $1 \leq j \leq k$ . Protocol  $\mathcal{P}$  repeatedly chooses an index  $1 \leq i \leq m$ , and executes one of these two actions, which can be implemented with some effort using only binary interactions: take  $k$  agents from  $\mathbf{x}_i$ , and dispatch them to  $\mathbf{y}_i^1, \dots, \mathbf{y}_i^k$  (one agent to each state); or take one agent from  $\mathbf{x}_i$  and  $(k-1)$  helpers, and dispatch them to  $\mathbf{z}_i^1, \dots, \mathbf{z}_i^k$ . If all agents of  $\mathbf{x}_i$  are dispatched for every  $1 \leq i \leq m$ , then we say that the *dispatch is correct*. Observe that a correct dispatch satisfies  $\mathbf{x} = k\mathbf{y} + \mathbf{z}$ .

The problem is that the dispatch may or may not be correct. Assume, e.g., that  $k = 5$  and  $m = 1$ . Consider the input  $x_1 = 17$ , and assume that  $\mathcal{P}$  has  $m \cdot (k-1)^2 = 16$  helpers.  $\mathcal{P}$  may correctly dispatch  $y_1 = 3$  agents to each of  $\mathbf{y}_1^1, \dots, \mathbf{y}_5^1$  and  $z_1 = 2$  to each of  $\mathbf{z}_1^1, \dots, \mathbf{z}_5^1$ ; this gives a total of  $(3+2) \cdot 5 = 25$  agents, consisting of the 17 agents for the input plus 8 helpers. However, it may also wrongly dispatch 2 agents to each  $\mathbf{y}_i^1$  and 4 agents to each of  $\mathbf{z}_i^1$ , with a total of  $(2+4) \cdot 5 = 30$  agents, consisting of 14 input agents plus 16 helpers. In the second case, each  $\tilde{\mathcal{P}}_j$  wrongly computes  $\tilde{\varphi}_j(2, 4) = \varphi_j(2 \cdot 5 + 4) = \varphi_j(14)$ , instead of the correct value  $\varphi_j(17)$ .

To solve this problem we ensure that  $\mathcal{P}$  can always recall agents already dispatched to  $\tilde{\mathcal{P}}_1, \dots, \tilde{\mathcal{P}}_k$  as long as the dispatch is not yet correct. This allows

$\mathcal{P}$  to “try out” dispatches until it dispatches correctly, which happens eventually w.p.1. For this we design  $\mathcal{P}$  so that the atomic protocols  $\tilde{\mathcal{P}}_1, \dots, \tilde{\mathcal{P}}_k$  can work with agents that arrive to the initial states over time (*dynamic initialization*), and can always return to their initial states and go back to  $\mathcal{P}$ , unless the dispatch is correct (*reversibility*). To ensure that  $\mathcal{P}$  stops recalling agents after a correct dispatch, we modify the dispatch transitions so that they become disabled when  $\mathbf{x}_1, \dots, \mathbf{x}_m$  are not populated.

## 4.2 State complexity: lower bounds

In [16,24] we have also studied the problem of obtaining lower bounds for  $STATE(\eta)$ . This question turns out to be surprisingly hard, and so we have focused on obtaining lower bounds for the state complexity of a particularly simple family of predicates, namely those of the form  $x \geq k$ . This amounts to finding a lower bound for the number  $n$  of states needed to decide  $x \geq k$ , or, equivalently, an upper bound for the largest number  $k$  such that  $x \geq k$  can be decided by a protocol with  $n$  states. We prefer the latter formulation due to its analogy with the busy beaver function. Recall that the busy beaver function assigns to a number  $n$  the largest  $\eta$  such that a Turing machine with at most  $n$  states, started on a blank tape, writes  $\eta$  consecutive 1s on the tape and halts. Analogously, the busy beaver function for population protocols assigns to  $n$  the largest  $\eta$  such that a population protocol with at most  $n$  states decides the predicate  $x \geq \eta$ . Intuitively,  $\eta$  is the largest number “recognizable” by protocols with at most  $n$  states.

We have obtained results for protocols with and without leaders. It is known that the time complexity of predicates is different for population protocols with and without leaders: While the first can decide any Presburger predicate in polylogarithmic parallel time [10], the latter need linear parallel time for majority [4]. Is the same true for state complexity? The question is still open, but we have made some progress.

Let  $BB, BB_L: \mathbb{N} \rightarrow \mathbb{N}$  be the busy beaver functions for leaderless protocols and for protocols with leaders, respectively. A protocol similar to the second one of Example 1, only simpler, decides  $x \geq 2^n$  with  $O(n)$  states, showing that  $BB(n) \in \Omega(2^n)$ . In [16] we prove  $BB_L(n) \in \Omega(2^{2^n})$ . This result is quite surprising: for certain numbers  $k$ , there are population protocols that decide  $x \geq k$  even though an agent does not have enough memory to index even one bit of  $k$ . The proof follows from a theorem by Mayr and Meyer on presentations of commutative semigroups [48], which can be reformulated in protocol terms as follows: for every  $n \geq 1$ , there exists a protocol with  $O(n)$  states and three distinguished states *start*, *end*, *counter* such that from an initial configuration that puts one agent in *start* and  $k$  agents in *counter*, respectively, it is possible to reach a configuration putting at least one agent in state *end* if and only if  $k \geq 2^{2^n}$ . By adding transitions that allow an agent in state *end* to attract all other agents to *end*, it is easy to obtain a protocol deciding  $x \geq 2^{2^n}$ .

Can we also obtain upper bounds on  $BB(n)$  and  $BB_L(n)$ , and so lower bounds on the state complexity? After some years investigating this question without

progress, we have recently made a breakthrough [24]. Our first result is that  $BB_L(n)$  is bounded by a variant of the Ackermann function. The result is proved by means of a pumping technique. We first show that if a protocol with  $n$  states answers 0 for two inputs  $a < b$  satisfying certain conditions, then it also answers 0 for every input  $a + \lambda(b - a)$ , and so the protocol cannot compute any predicate of the form  $x \geq \eta$ . Then we find a function  $F(n)$  such that if a protocol with at most  $n$  states rejects all inputs with at most  $F(n)$  agents, then there are inputs  $a < b < F(n)$  satisfying the conditions. The function is obtained using results from the theory of controlled sequences, an area of mathematics related to well-quasi-orders [2,12,37].

An Ackermannian upper bound may seem extremely weak. However, it follows from recent results in the theory of Petri nets that functions similar to  $BB_L(n)$  have an Ackermannian *lower* bound. To give an example, say that a protocol *weakly decides* the predicate  $x \geq k$  if the following holds: for every initial configuration with at least  $k$  agents there exists a run leading to a configuration with consensus 1; for every initial configuration with less than  $k$  agents, no such configuration is reachable. Then the largest  $k$  such that  $x \geq k$  is weakly computable with  $n$  states is an Ackermannian function of  $n$ .

The main result of [24] is a triple exponential bound on  $BB(n)$ . That is, leaderless protocols with at most  $n$  states can recognize numbers at most triple exponential in  $n$ . The proof technique is again a pumping lemma. The key property of leaderless protocols that we use to obtain an elementary bound is that, loosely speaking, the set of initial configurations of a leaderless protocol is closed under addition. To understand this, observe that initial configurations of a leaderless protocol deciding  $x \geq k$  put  $k$  agents in the initial state and 0 agents in all others. Therefore, the sum of two initial configurations with  $k_1$  and  $k_2$  agents is the initial configuration with  $k_1 + k_2$  agents. This does no longer hold for protocols with a leader, whose initial configurations also put one agent in the initial state of the leader; in this case, the sum of two initial configurations with a leader is a configuration with *two* leaders.

In an unpublished result obtained together with Jérôme Leroux, we have improved the bound for leaderless protocols to double exponential; we conjecture that the optimal upper bound is single exponential, matching the lower bound. But currently we do not even have a line of attack to obtain an elementary bound for protocols with a leader.

## 5 Conclusions and future work

We have surveyed our recent work on the verification of population protocols, and on their state complexity. This work has produced PEREGRINE, the first automatic tool able to verify correctness of protocols for all inputs. In the verification area, there are many open directions for future work:

- Protocols are often designed parametrically, for example, one gives a construction that yields a protocol deciding  $ax - by \geq c$  for arbitrary coefficients

$a, b, c$ . Our methods cannot yet prove that the construction is correct for every  $a, b, c$ . It is easy to show that verifying infinite families of protocols is an undecidable problem, even for very restricted cases, but it should be possible to design procedures that perform well in practice.

- As mentioned in the introduction, in the last years families of protocols that decide one single predicate, but where the number of states increases with the number of agents, have been intensely investigated, see e.g. [5,27]. Again, we do not have any verification technique for them.
- The work initiated in [19] on the automatic verification of the expected runtime of a protocol is still in its infancy.

Our work on the state complexity problem is tightly linked to difficult problems of the theory of Petri nets. The obvious future direction is closing the current gaps between the upper and lower bounds for the busy beaver functions in the leaderless case, and the case with leaders. We consider this a fundamental problem in the theory of population protocols. Intuitively, it measures quantitatively the relation between the microscopic scale of agents and the macroscopic scale of the predicates they decide.

## References

1. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.: General decidability theorems for infinite-state systems. In: LICS. pp. 313–321. IEEE Computer Society (1996)
2. Abriola, S., Figueira, S., Senno, G.: Linearizing well quasi-orders and bounding the length of bad sequences. *Theor. Comput. Sci.* **603**, 3–22 (2015)
3. Alistarh, D., Aspnes, J., Eisenstat, D., Gelashvili, R., Rivest, R.L.: Time-space trade-offs in population protocols. In: SODA. pp. 2560–2579. SIAM (2017)
4. Alistarh, D., Aspnes, J., Gelashvili, R.: Space-optimal majority in population protocols. In: SODA. pp. 2221–2239. SIAM (2018)
5. Alistarh, D., Gelashvili, R.: Recent algorithmic advances in population protocols. *SIGACT News* **49**(3), 63–73 (2018)
6. Alistarh, D., Gelashvili, R., Vojnovic, M.: Fast and exact majority in population protocols. In: PODC. pp. 47–56. ACM (2015)
7. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. In: PODC. pp. 290–299. ACM (2004)
8. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. *Distributed Comput.* **18**(4), 235–253 (2006)
9. Angluin, D., Aspnes, J., Eisenstat, D.: Stably computable predicates are semilinear. In: PODC. pp. 292–299. ACM (2006)
10. Angluin, D., Aspnes, J., Eisenstat, D.: Fast computation by population protocols with a leader. *Distributed Computing* **21**(3), 183–199 (2008)
11. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. *Distributed Computing* **20**(4), 279–304 (2007)
12. Balasubramanian, A.R.: Complexity of controlled bad sequences over finite sets of  $\mathbb{N}^d$ . In: LICS. pp. 130–140. ACM (2020)

13. Blondin, M.: The ABCs of Petri net reachability relaxations. *ACM SIGLOG News* **7**(3), 29–43 (2020)
14. Blondin, M., Esparza, J., Genest, B., Helfrich, M., Jaax, S.: Succinct population protocols for presburger arithmetic. In: STACS. LIPIcs, vol. 154, pp. 40:1–40:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
15. Blondin, M., Esparza, J., Helfrich, M., Kucera, A., Meyer, P.J.: Checking qualitative liveness properties of replicated systems with stochastic scheduling. In: CAV (2). Lecture Notes in Computer Science, vol. 12225, pp. 372–397. Springer (2020)
16. Blondin, M., Esparza, J., Jaax, S.: Large flocks of small birds: on the minimal size of population protocols. In: STACS. LIPIcs, vol. 96, pp. 16:1–16:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018)
17. Blondin, M., Esparza, J., Jaax, S.: Peregrine: A tool for the analysis of population protocols. In: CAV (1). Lecture Notes in Computer Science, vol. 10981, pp. 604–611. Springer (2018)
18. Blondin, M., Esparza, J., Jaax, S., Meyer, P.J.: Towards efficient verification of population protocols. In: PODC. pp. 423–430. ACM (2017)
19. Blondin, M., Esparza, J., Kucera, A.: Automatic analysis of expected termination time for population protocols. In: CONCUR. LIPIcs, vol. 118, pp. 33:1–33:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018)
20. Blondin, M., Finkel, A., Haase, C., Haddad, S.: The logical view on continuous Petri nets. *ACM Transactions on Computational Logic (TOCL)* **18**(3), 24:1–24:28 (2017)
21. Cardelli, L., Csikász-Nagy, A.: The cell cycle switch computes approximate majority. *Scientific Reports* **2**(1), 656 (2012)
22. Chatzigiannakis, I., Michail, O., Spirakis, P.G.: Algorithmic verification of population protocols. In: SSS. Lecture Notes in Computer Science, vol. 6366, pp. 221–235. Springer (2010)
23. 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)
24. Czermer, P., Esparza, J.: Lower bounds on the state complexity of population protocols. In: PODC. pp. 45–54. ACM (2021)
25. Czerwinski, W., Lasota, S., Lazic, R., Leroux, J., Mazowiecki, F.: The reachability problem for Petri nets is not elementary. *J. ACM* **68**(1), 7:1–7:28 (2021)
26. Czerwinski, W., Orlikowski, L.: Reachability in vector addition systems is Ackermann-complete. *CoRR* **abs/2104.13866** (2021)
27. Elsässer, R., Radzik, T.: Recent results in population protocols for exact majority and leader election. *Bulletin of the EATCS* **126** (2018)
28. Esparza, J.: Advances in parameterized verification of population protocols. In: CSR. Lecture Notes in Computer Science, vol. 10304, pp. 7–14. Springer (2017)
29. Esparza, J., Gaiser, A., Kiefer, S.: Proving termination of probabilistic programs using patterns. In: CAV. Lecture Notes in Computer Science, vol. 7358, pp. 123–138. Springer (2012)
30. Esparza, J., Ganty, P., Leroux, J., Majumdar, R.: Model checking population protocols. In: FSTTCS. LIPIcs, vol. 65, pp. 27:1–27:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016)
31. Esparza, J., Ganty, P., Leroux, J., Majumdar, R.: Verification of population protocols. *Acta Informatica* **54**(2), 191–215 (2017)
32. Esparza, J., Helfrich, M., Jaax, S., Meyer, P.J.: Peregrine 2.0: Explaining correctness of population protocols through stage graphs. In: ATVA. Lecture Notes in Computer Science, vol. 12302, pp. 550–556. Springer (2020)



33. Esparza, J., Jaax, S., Raskin, M.A., Weil-Kennedy, C.: The complexity of verifying population protocols. *Distributed Comput.* **34**(2), 133–177 (2021)
34. Esparza, J., Ledesma-Garza, R., Majumdar, R., Meyer, P.J., Niksic, F.: An SMT-based approach to coverability analysis. In: *CAV. Lecture Notes in Computer Science*, vol. 8559, pp. 603–619. Springer (2014)
35. Esparza, J., Melzer, S.: Verification of safety properties using integer programming: Beyond the state equation. *Formal Methods Syst. Des.* **16**(2), 159–189 (2000)
36. Esparza, J., Raskin, M.A., Weil-Kennedy, C.: Parameterized analysis of immediate observation Petri nets. In: *Petri Nets. Lecture Notes in Computer Science*, vol. 11522, pp. 365–385. Springer (2019)
37. Figueira, D., Figueira, S., Schmitz, S., Schnoebelen, P.: Ackermannian and primitive-recursive bounds with Dickson’s lemma. In: *LICS*. pp. 269–278. IEEE Computer Society (2011)
38. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! *Theoretical Computer Science* **256**(1-2), 63–92 (2001)
39. Fraca, E., Haddad, S.: Complexity analysis of continuous Petri nets. *Fundam. Informaticae* **137**(1), 1–28 (2015)
40. Gasieniec, L., Stachowiak, G.: Enhanced phase clocks, population protocols, and fast space optimal leader election. *J. ACM* **68**(1), 2:1–2:21 (2021)
41. Haase, C.: A survival guide to Presburger arithmetic. *SIGLOG News* **5**(3), 67–82 (2018)
42. Jancar, P., Purser, D.: Structural liveness of Petri nets is EXPSPACE-hard and decidable. *Acta Informatica* **56**(6), 537–552 (2019)
43. Kosaraju, S.R.: Decidability of reachability in vector addition systems (preliminary version). In: *STOC*. pp. 267–281. ACM (1982)
44. Lengál, O., Lin, A.W., Majumdar, R., Rümmer, P.: Fair termination for parameterized probabilistic concurrent systems. In: *TACAS* (1). *Lecture Notes in Computer Science*, vol. 10205, pp. 499–517 (2017)
45. Leroux, J.: Vector addition systems reachability problem (A simpler solution). In: *Turing-100. EPiC Series in Computing*, vol. 10, pp. 214–228. EasyChair (2012)
46. Leroux, J.: The reachability problem for Petri nets is not primitive recursive. *CoRR abs/2104.12695* (2021)
47. Mayr, E.W.: An algorithm for the general Petri net reachability problem. In: *STOC*. pp. 238–246. ACM (1981)
48. Meyer, E.W., Meyer, A.R.: The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in mathematics* **46**(3), 305–329 (1982)
49. Moran, P.A.P.: Random processes in genetics. *Mathematical Proceedings of the Cambridge Philosophical Society* **54**(1), 60–71 (1958)
50. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: *TACAS. Lecture Notes in Computer Science*, vol. 4963, pp. 337–340. Springer (2008)
51. Murata, T.: Petri nets: Properties, analysis and applications. *Proceedings of the IEEE* **77**(4), 541–580 (1989)
52. Reisig, W.: *Petri nets: An introduction*, EATCS Monographs on Theoretical Computer Science, vol. 4. Springer (1985)
53. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. *Natural Computing* **7**(4), 615–633 (2008)