Distributed Population Protocols: Naturally!

David de Frutos Escrig

Dpto. Sistemas Informáticos y Computación, Facultad de Ciencias Matemáticas
Universidad Complutense de Madrid, Madrid, Spain
defrutos@sip.ucm.es

Abstract. Classical population protocols manage a fix size population of agents that are created by the input population: one agent exactly per unit of the input. As a consequence, complex protocols that have to perform several independent tasks find here a clear bottleneck that drastically reduces the parallelism in the execution of those tasks. To solve this problem, I propose to manage distributed population protocols, that simply generalize the classical ones by generating a (fix, finite) set of agents per each unit of the input. A surprising fact is that these protocols are not really new, if instead of considering only the classical protocols with an input alphabet we consider the alternative simpler one that states as input mechanism a given subset of the set of states. Distributed population protocols are not only interesting because they allow more parallel and faster executions, but specially because the distribution of both code and data will allow much simpler protocols, inspired by the distribution of both places and transitions in Petri nets.

1 Introduction

Population protocols were introduced by Angluin et al. [4, 5] as a new proposal for distributed computation by very limited agents, whose computational power is based on the interactions between them. These agents are supposed *to be moving* in an uncontrolled way in a common arena, so that from time to time they meet one another. In their meetings they communicate each other their state, after which they can change it according to the received information. The initial formalization is extremely simple, reflecting all the ingredients in the description above, and producing a formalism very easy to use. But at the same time, their creators foresaw that the main idea could be used in more liberal ways, and so they also introduced a general framework allowing many generalizations. Indeed, several of them have been considered interesting by several groups of researchers during the last years, and many definitions of new classes of population protocols have appeared, and their computational power and characteristics have been studied in detail. You can find in [18] a nice very compact informative essay.

Probably, the first (intended) limitation of the original model was the conservation of the number of agents working in the protocol. This is justified by a possible hardware implementation of those agents as *nanocomputers*. Definitely, this restriction, together with the fix limitation of memory of each agent, is the main reason of the quite limited computational power of the classical population protocols. Their creators characterized it as that of *semilinear predicates*, or equivalently those definable in *Presburger arithmetic* [6, 8]. The *input* component at the definition establishes a (finite) input alphabet, and the multisets on it are the concrete inputs. Besides, each unit of this input creates one agent (in a certain initial state) and thus the number of agents all along the execution of the protocol coincides with the size of the input.

Another main characteristic of the classical population protocols is that the protocol itself (its code) is *uniform* and does not know in advance the set of agents that is working on it. This *code* is just a list of transitions in which one agent can participate when meeting with another; the same list for all the agents. Finally, the agents are assumed to be (and they are indeed, by definition) only differentiable by their state, so that there are no unique identifiers for them from scratch. But fairness of the executions is the crucial assumption in order to get their computational power. Fairness filters out the executions that are considered to be feasible, and then the protocols can be designed in a way that all the fair executions from the same input lead all the agents working at it to the same output, that is the output computed for that input.

Another justification of fairness arrives when we consider the *uniform* probabilistic selection of the pair of agents that communicate at each step of an execution. It is well known that in bounded non-deterministic models, where the probabilities of the next step to be executed are fixed, fairness is equivalent to *probability 1 computation*, so that we can neglect the consideration of unfair executions since its full set has probability 0 . The introduction of probabilities also allows the computation of the expected time for stabilization of a protocol, which provides a natural measure of the *efficiency* of population protocols.

By ruling out the imposed restrictions much more powerful classes of protocols have been introduced. Doty et al. have developed a large collection of papers where the dynamic creation of agents makes possible the delay of certain transitions, simply because a few pairs of agents can participate in them. Under these assumptions fairness and probability 1 are not equivalent anymore, and this makes the probabilistic model much more powerful, even exceeding the power of Turing machines (of course in a *non-implementable* way) [13]. And by extending the memory of agents, which will be related with the size of the input, Alistarh et al. [1] have provided protocols that compute more predicates, studying a trade-off between used memory and needed time for stabilization, similar to the existing one in classical computation. Finally, by removing the (total) symmetry, we can get either faster protocols or again a bigger computation power. Protocols with leaders [7, 15] and agents with unique identification [17] work in those directions. More about recent extensions of the basic model can be found in [2].

In this paper I present a new methodology for the design of protocols in the basic class, that advocates for (even) more distributed protocols. It is obtained by removing the one to one correspondence between the units of the input and the agents participating in the computation of the protocol. Instead, the input function will introduce at the initial configuration a certain collection of agents at some preset states. After that they evolve following the rules in the classical definition.

I want to stress the fact that I am not introducing *another* extension of the basic model, but just a new methodology to define protocols that remain at that basic model. What I am claiming is that the use of (mainly only) classical protocols is constraining the designs of people developing basic protocols, so that they are forced to use quite complicated techniques to construct their protocols (see for instance [9]). In particular, I will show that the use of my distributed population protocols have led me to obtain protocols where *reverse* transitions, which reverse the effect of others, are totally removed, or their role is quite limited. These reverse transitions seemed unavoidable to maintain the state space of the protocols small, but we will see that the distribution of tasks will facilitate alternative (and very natural) solutions that do not need (or reduce a lot) their use. Reverse transitions will always delay the stabilization of the protocols, and in most of the cases they will do it in an absolutely unacceptable (overexponential) way. Thus their removal is a clear improvement.

The rest of the paper is organized as follows. In the next section the basic notation is presented and the definitions of (classical) population protocols recalled, explaining how I 'derived' the one for distributed protocols, starting from them. In Section 3, I give the formal definition of distributed population protocol, and the protocol $\mathcal{P}_\wedge$ for the computation of $\phi = \phi_1 \wedge \phi_2$ is introduced as a representative example. Next, in Section 4, I present a different application of distributed protocols, based in the 2-base representation of numbers, instead of using the *unary* representation, as usually done when developing population protocols. This representation is also used in Section 5, where I present a *succinct* protocol for the computation of threshold predicates that appear at the 'basis' generating the Presburger predicates. I conclude collecting the main goals reached at the paper, and announcing several continuations.

Some simple proofs have been omitted, and others sketched.

2 Preliminaries

Throughout the paper I use the standard mathematical notation. In particular, $\mathbb{Z}$ denotes the set of integers, and $\mathbb{N}$ the set of natural numbers (i.e., non-negative integers). I use $a..b$ with $a \leq b$ to denote an interval in any of those sets, and A^B for the set of (total) functions from B to A, so that $A^{1..k}$ are the *vectors* with k elements in A. Also, (finite) *multisets* over a finite set X are the elements of $\mathbb{N}^X$. The size of $\alpha \in \mathbb{N}^X$ is defined as $|\alpha| = \sum_{x \in X} \alpha(x)$. When the elements of X are summable, we define $\sum_{q \in \alpha} q = \sum_{x \in X} \alpha(x) \times x$. We say that $x \in X$ is in a multiset α, and write $x \in \alpha$ by abuse of notation, if $\alpha(x) > 0$. Multisets

will be represented either by extension, in the form $\{\{x_1, \ldots, x_k\}\}$ or in a linear way $k_1 \cdot x_1, \ldots, k_l \cdot x_l$, where the elements x_i will be different one another, and $k_i > 0$. $\alpha + \beta$ denotes multiset sum, which means $(\alpha + \beta)(x) = \alpha(x) + \beta(x)$, for $x \in X$. Besides, $\alpha \leq \beta$ means $\alpha(x) \leq \beta(x)$ for all $x \in X$, and when $\beta \leq \alpha$, $\alpha - \beta$ denotes the multiset difference, $(\alpha - \beta)(x) = \alpha(x) - \beta(x)$. Finally, for $k \in \mathbb{N}$ and $\alpha \in \mathbb{N}^X$, we define the product $k \times \alpha$ as the multiset given by $(k \times \alpha)(x) = k\,\alpha(x)$, for each $x \in X$. The empty multiset is denoted by $\mathbf{0}$.

From classical population protocols to distributed protocols. A *population protocol* (as in [5]) is a tuple $\mathcal{P} = (Q, I, T, O)$, where $I : X \to Q$, for some *input alphabet* X, is the *input initialization* function, $T \subseteq Q^2 \times Q^2$, and $O : Q \to Y$ is the *output* function, for some *output alphabet* Y. Transitions will be usually presented in the form $q_1, q_2 \to q_1', q_2'$. We assume that T is total: for all $(q_1, q_2) \in Q^2$ there exists (q_1', q_2') with $q_1, q_2 \to q_1', q_2'$. Whenever this is not originally the case, the set of transitions is completed with the necessary identity transitions $q_1, q_2 \to q_1, q_2$, that are said to be *silent*. In this paper we only consider protocols that compute a predicate, which corresponds to $Y = \{0, 1\}$.

An alternative slightly different formalization used in many papers, even by their original creators (e.g. in [6],) removes the input alphabet X, taking directly $I \subseteq Q$. This means that there is a subset of *initial states*, so that concrete inputs are just the multisets in $\mathbb{N}^I$. In order to distinguish both definitions, I will call *pure* population protocols to those defined using this alternative formalization.

The semantics of population protocols is the same in both cases. A configuration is any multiset in $\mathbb{N}^Q$. We say that a transition $q_1, q_2 \to q_1', q_2'$ is *enabled* in a configuration C, when $\{\{q_1, q_2\}\} \leq C$. Then, it can be *fired,* leading to $C' = C - \{\{q_1, q_2\}\} + \{\{q_1', q_2'\}\}$, and we write $C \to C'$. An execution of $\mathcal{P}$ is an infinite sequence $\sigma = C_0 \to C_1 \to C_2 \ldots$. The output of a non-empty configuration C is defined by $O(C) = b$ if and only if for all $q \in C$ we have $O(q) = b$. Then, we say that an execution σ has output $O(\sigma) = b$, if there exists $i \in \mathbb{N}$ such that for all $c \in \mathbb{N}$ we have $O(C_{i+c}) = b$. However, not all executions count, we only consider *fair* computations: σ is fair if
$\forall D \in \mathbb{N}^Q \;\; (\,|\,\{i \in \mathbb{N} : C_i \xrightarrow{*} D\,\}\,| = \infty \implies |\,\{j \in \mathbb{N} : C_j = D\}\,| = \infty\,)$.

Now we say that a *classical* population protocol $\mathcal{P}$ is *well-specified* and computes the predicate $\phi_{\mathcal{P}}$ on any *input population* $\alpha \in \mathbb{N}^X$, if for any fair execution σ starting at the initial configuration $C_0 = I(\alpha)$ we have $\phi_{\mathcal{P}}(\alpha) = O(\sigma)$. Instead, a *pure* population protocol $\mathcal{P}$ may compute a predicate on the set of *input configurations* $\mathbb{N}^I$: $\mathcal{P}$ indeed computes $\phi_{\mathcal{P}}$, if for any input configuration C_0, and any fair execution σ starting at C_0, we have $\phi_{\mathcal{P}}(C_0) = O(\sigma)$.

So, the only difference between these two formalisms is the way the input is presented: Ordinary protocols take multisets $\alpha \in \mathbb{N}^X$ that are converted into $I(\alpha) \in \mathbb{N}^X$: $I(\alpha)(q) = \sum_{I(x)=q} \alpha(x)$; while in the second case we have directly any multiset $C_0 \in \mathbb{N}^I$ as input. It is true that taking $X = I \subseteq Q$ and the *identity* function as input function we can see any pure protocol as a classical protocol, but this requires the use of input alphabets more elaborate that the usually considered in the classical protocols. However, whenever we are defining the expressive power of a formalism we cannot assume/impose any 'natural'

presentation of the input: we cannot state that some states cannot be taken as initial. Following this idea, when the expressive power of population protocols was established their creators considered in [6] pure and not classical ones, although the 'emulation' result above proves that classical protocols have the same expressive power. Even so, it is true that the use of the external input alphabet looks as very suggestive, since it allows the separation of the description of the input of the solved problem from the details (the set of states) of the protocol that will try to solve it. Nevertheless, if we insist on the use of this input alphabet, sometimes this could turn into a too restrictive constraint, instead of being a guide for the definition of the desired protocols.

Next I will try to motivate my distributed protocols seeing that the use of the input alphabet by itself is not the root of the problem. But we need to loosen up the one to one relation between the units of the input and the agents of the protocol if we want to design simple and powerful protocols in an easy way.

Let us recall how it is proved [5, 12, 9] that the class of predicates definable by some class of protocols is *algebraically closed* with respect to Boolean operations. This is done considering classical and not pure protocols. In particular, given a fix input alphabet $X = \{x_1, ..., x_k\}$, and a pair of protocols $\mathcal{P}_1, \mathcal{P}_2$ computing $\phi_1(x_1, ..., x_k)$ and $\phi_2(x_1, ..., x_k)$, a protocol $\mathcal{P}_\wedge$ computing $\phi_1 \wedge \phi_2$ is constructed. The first problem with this approach is that whenever we talk about a predicate we typically assume a certain 'universe' (the set of variables of which 'it depends') where it is defined. We need to prove that the conjunction of two (arbitrary) definable predicates is also definable. But then they could 'depend' on two different sets of variables ... Yes, no (formal) problem, we can always (although artificially) enlarge the set of variables on which a predicate (formally) depends, but if we do this in this framework, it will not be for free! The natural implementation of each predicate will only receive as inputs those corresponding to the value of the variables it depends, but if now we enlarge the input alphabet there will be input units for all the variables on which **either** ϕ_1 or ϕ_2 depends. Then, both protocols will receive the agents generated by all the input units, even if those corresponding to the variables that are not needed in one of them should not play any role in its computation ... But now they must collaborate in it, and in particular they must receive the final output when it is computed ... We are obliged by the constraint introduced by the use of classical protocols to proceed this way. It seems really strange that we assume these costs, at least without any discussion. At least it is easy to incorporate the additional input agents in the two protocols as *dummy* agents, that are only there, only waiting for the reception of the computed output value, in order to obtain the required total stable consensus. Since at none of the papers cited above there is a single word about this annoying technical problem, I cannot speculate about whether they did not notice it or they considered the costs perfectly acceptable. But for me this was a strong motivation for looking for another more natural solution where we will need not to invest on totally useless agents.

In particular, if ϕ_1 and ϕ_2 depend on two disjoint sets of variables, e.g. we have $\phi_1(x_1, ..., x_k), \phi_2(y_1, ..., y_l)$, then we can avoid the introduction of all the

dummy agents simply considering $\mathcal{P}_1$ working on X and $\mathcal{P}_2$ working on Y. In this way we obtain $\mathcal{P}_\wedge$ based on the *aggregation* of these two (sub)protocols: we only need a final phase for the calculation and dissemination of the conjunction of the values computed by them. Quite natural, quite easy. And definitely exploding the natural distributed character of population protocols: there were two independent tasks to solve and we have distributed the needed inputs to do it in a totally independent way.

However, this solution seems impossible whenever the two predicates 'really depend' on the same set of variables X. Let us assume that we have now $\phi_1(x_1, ..., x_k)$ and $\phi_2(x_1, ..., x_k)$. Even if the states in $\mathcal{P}_1$ and $\mathcal{P}_2$ remain disjoint, we cannot define $I : X \to Q_1 \cup Q_2$ in a satisfactory way as above, since by definition we must direct all the agents corresponding to the same variable to the same state, and this can only correspond to either Q_1 or Q_2 . But the problem disappears if we consider pure population protocols instead of classical ones. If we need to combine $\mathcal{P}_1 = (Q_1, T_1, I_1, O_1)$ and $\mathcal{P}_2 = (Q_2, T_2, I_2, O_2)$, we can assume that Q_1 and Q_2 are disjoint, and then ... I_1 and I_2 are too! This means that we can define $\mathcal{P}_\wedge$ as above, with Q based on the union of the states of Q_1 and Q_2, taking $I = I_1 \cup I_2$, and **everything works fine!** Definitely, in the pure framework there is no reason for setting aside that simple definition of $\mathcal{P}_\wedge$. Free of the constraint of the input alphabet, we have seen that pure protocols are more flexible (although formally remain equivalent) that classical protocols. Why should we insist on working 'mainly' with the latter?

We can interpret the solution above in two different ways: the 'algebraic interpretation' *renames* the variables in ϕ_2, so that now the variables in both components are disjoint, and then we can proceed as in our introductory example. The second interpretation recovers the role of the (common) input alphabet X , that has been hidden by our pure protocols. What we have done can be understood as a *replication* of the (natural) input, so that both (sub)protocols will receive a separate copy of it. We have obtained two independent collection of agents that will compute $\phi_1(x_1, ..., x_k)$ and $\phi_2(x_1, ..., x_k)$ in a distributed way.

Let me now reply to a possible criticism at this point: my protocol $\mathcal{P}_\wedge$ when considered in the *pure* protocols framework is not exactly computing $\phi(x_1, ..., x_k) = \phi_1(x_1, ..., x_k) \wedge \phi_2(x_1, ..., x_k)$, but ... **more than this!** This is certainly true, but who cares? Indeed, when taking $I = I_1 \cup I_2$ we are stating that we can use the states in both subsets as (independent) initial states for the agents participating in $\mathcal{P}_\wedge$. As a consequence, what we have exactly constructed is a protocol for the (general) computation of $\overline{\phi}(x_1, ..., x_k, y_1, ..., y_k) = \phi_1(x_1, ..., x_k) \wedge \phi_2(y_1, ..., y_k)$, since we can inject a different number of agents at the two states 'representing' each variable in X , at the two sets I_1 and I_2. But we can perfectly claim that this more general protocol is also an implementation of our desired predicate $\phi(x_1, \ldots, x_k) = \phi(x_1, \ldots, x_k) \wedge \phi_2(x_1, ..., x_k)$, simply injecting the same number of agents x_i at the two states representing that argument in I_1 and I_2, *that's all.*

Instead, when in [5, 12, 9] the corresponding protocol $\mathcal{P}_\wedge$ is defined, they need to use $Q = Q_1 \times Q_2$, so that the pure version of the protocol would also take

$I = I_1 \times I_2$, since in order to preserve the one to one relation between the units defining the values of the variables in X and the agents of $\mathcal{P}_\wedge$ they need to *pair* the 'subagents' working in $\mathcal{P}_1$ and $\mathcal{P}_2$ using the notion of *parallel composition*. This produces a **less distributed** protocol and besides the pairing between the subagents does not reflect any logical connection between the two components, that could be paired in a totally arbitrary way.

Let us conclude observing that the proposal above is also exploting the idea of population splitting, as in [2], although I got to it following a different path. Although in an implicit way, I am using a *typing* mechanism that separates the agents working in a protocol in several disjoint *classes* in such a way that the code of the protocol will preserve the *type* of the agents participating in each transition. In this way we get a safe modular methodology (for free!), in the sense *typed programming languages* provide.

3 Distributed population protocols

Definition 1. *A* distributed population protocol *is a tuple* $\mathcal{P} = (Q, I, T, O)$, *where* $I : X \to \mathbb{N}^Q$, *for some* input alphabet $X = \{x_1, \ldots, x_m\}$, *is the* input initialization *function,* $T \subseteq Q^2 \times Q^2$ *and* $O : Q \to \{0, 1\}$ *is the* output *function.*

Transitions are presented as for ordinary protocols. Since we only consider protocols that (possibly) compute a predicate, we have 0 and 1 as output values. Distributed population protocols compute as the ordinary ones, but considering the executions that start at the initial configurations $I(\alpha)$, where $\alpha \in \mathbb{N}^X$ is the input population: $I(\alpha) = \sum_{i=1}^{m} \alpha(x_i) \times I(x_i)$.

Next, I give the definition of a simple distributed protocol computing the *conjunction* of $\phi_1(x_1, ..., x_k)$ and $\phi_2(x_1, ..., x_k)$, computed by two distributed population protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ that work on the same input alphabet X.

Definition 2. *Given two distributed population protocols* $\mathcal{P}_1 = (Q_1, I_1, T_1, O_1)$ *and* $\mathcal{P}_2 = (Q_2, I_2, T_2, O_2)$ *that work on the input alphabet* $X = \{x_1, \ldots, x_k\}$ *and compute* $\phi_1(x_1, ..., x_k)$ *and* $\phi_2(x_1, ..., x_k)$, *we define* $\mathcal{P}_\wedge = (Q, I, T, O)$, *taking* $Q = (Q_1 \dot\cup Q_2) \times \{0, 1\}, I(x) = I_1(x) \times \{0\} + I_2(x) \times \{0\}$, $O((q_i, b)) = b$, *and* T *containing the* extended *transitions*

$$(q_{i,1}, q_{i,2} \to q'_{i,1}, q'_{i,2}) \in T_i \implies ((q_{i,1}, b_1), (q_{i,2}, b_2) \to (q'_{i,1}, b_1), (q'_{i,2}, b_2)) \in T$$

and the communication transitions

$$(q_{1,1}, b_1), (q_{2,1}, b_2) \to (q_{1,1}, O_1(q_{1,1}) \wedge O_2(q_{2,1})), (q_{2,1}, O_1(q_{1,1}) \wedge O_2(q_{2,1}))$$

where we are denoting by $q_{i,j}$ *the states in* Q_i .

Theorem 1. *Given two distributed population protocols* $\mathcal{P}_1 = (Q_1, I_1, T_1, O_1)$ *and* $\mathcal{P}_2 = (Q_2, I_2, T_2, O_2)$, *that work on the input alphabet* $X = \{x_1, \ldots, x_k\}$, *and compute the predicates* $\phi_1(x_1, ..., x_k)$ *and* $\phi_2(x_1, ..., x_k)$, *the protocol* $\mathcal{P}_\wedge = (Q, I, T, O)$ *defined in Definition 2 computes the predicate* $\phi = \phi_1 \wedge \phi_2$.

Proof. We decompose the protocol in two layers, as defined in [12], separating the extended and the communication transitions. The former work on the first component of the states in Q exactly as the transitions of $\mathcal{P}_1$ or $\mathcal{P}_2$, so that any fair execution of the extended transitions will lead to a configuration $C = C_1 + C_2$, where each C_i corresponds to one stable configuration of $\mathcal{P}_i$, with $O(C_i) = \phi_i(\overline{x})$. Next the communication transitions will produce the value $\phi(\overline{x})$ at all the agents, getting a *stable* configuration preserving this output. □

Once we have defined the conjunction and disjunction of two well-specified distributed protocols, an immediate induction allows us to compute any Boolean combination of a collection of protocols $\mathcal{P}_1, \ldots, \mathcal{P}_k$. I only consider expressions e with conjunction and disjunction as operators, since negation can be pushed to the leaves of the expression by applying De Morgan's laws, and negation of a protocol is obtained simply interchanging 0 and 1 at its output function. Note that the list of protocols above corresponds to all the occurrences of the protocols at the k leaves of the expression, even if this may suppose repeated occurrences of some protocols. Therefore, the expression contains exactly $k-1$ binary operators. We need to proceed this way since the repeated application of Definition 2 cannot take any advantage of repeated occurrences of the protocols.

Let us see which are the states of the protocol $\mathcal{P}_e$ that computes the value of e, having $\mathcal{P}_1, \ldots, \mathcal{P}_k$ as leaves. As usual, we can label each protocol argument with the binary path pt_i from the root of the expression to it. We denote by d_i the length of pt_i. We also denote by $\odot_{pt}$ the operator at any internal node reached by pt. Then the protocol $\mathcal{P}_e$ obtained by iterated application of Definition 2 to the subexpressions of e can be explicitly described as follows:

Definition 3. *Given a tree Boolean expression e having the well-specified distributed protocols $\mathcal{P}_1, \ldots, \mathcal{P}_k$ at its k leaves, we define $\mathcal{P}^e = (Q^e, I^e, T^e, O^e)$ taking $Q^e = \cup_{i=1}^k (Q_i^{pt_i} \times \{0,1\}^{d_i})$; $I^e(x) = \sum_{i=1}^k I_i(x)^0$, where ^{0} adds d_i 0's to obtain states in Q^e; $O((q_i, s)) = s[1]$; and T^e contains the* extended *transitions*

$$(q^i_{j_1}, q^i_{j_2} \to q^i_{j'_1}, q^i_{j'_2}) \in T_i \implies ((q^i_{j_1}, s_1), (q^i_{j_2}, s_2) \to (q^i_{j'_1}, s_1), (q^i_{j'_2}, s_2)) \in T^e$$

and the communication transitions $((q^i_{j_1}, s_1), (q^{i'}_{j_2}, s_2) \to ((q^i_{j_1}, s'_1), (q^{i'}_{j_2}, s'_2)$, *where $i \neq i'$, we are denoting by q^i_j the states in Q_i, and s'_1 and s'_2 are obtained from s_1 and s_2 by considering the longest common prefix $pt_{i,i'}$ of pt_i and pt'_i and its length $lcp_{i,i'}$, simply turning the $(lcp_{i,i'}+1)$-th bit of them into $v_1 \odot_{pt_{i,i'}} v_2$ where $v_1 = s_1[lcp_{i,i'}+2]$ when $lcp_{i,i'}+1 \neq d_i$, and $v_1 = O_1(q^i_{j_1})$, otherwise; and analogously $v_2 = s_2[lcp_{i,i'}+2]$ when $lcp_{i,i'}+1 \neq d_{i'}$, and $v_2 = O_2(q^{i'}_{j_2})$, otherwise.*

A couple of examples may clarify this notation. First, we consider i with $pt_i = (1,1,0,1)$ and i' with $pt'_i = (1,1,1,0,0)$, so that $pt_{i,i'} = (1,1)$ and $lcp_{i,i'} = 2$. Then, if $s_1 = (0,1,0,0)$, $s_2 = (1,0,1,1,0)$ and $\odot_{pt_{i,i'}} = \wedge$, we obtain $s'_1 = (0,1,0,0)$ and $s'_2 = (1,0,0,1,0)$, since $s_1[4] \wedge s_2[4] = 0 \wedge 1 = 0$. While if $pt'_i = (1,1,1)$ and $s_2 = (1,0,1)$ we need to consider $O_{i'}(q^{i'}_{j_2})$, obtaining $s_1[4] \wedge O_{i'}(q^{i'}_{j_2}) = 0 \wedge O_{i'}(q^{i'}_{j_2}) = 0$, which produces $s'_1 = (0,1,0,0)$ and $s'_2 = (1,0,0)$.

Although an immediate inductive reasoning provides the correctness proof of this protocol, if we analyze it in a global way we can observe a layered structure,

with as many layers as the height of e. At the first layer the computation of all the argument protocols is done (possibly decomposed into sublayers, depending on the characteristics of each one of them), and then the following layers contain the computations of the subexpressions rooted at each internal node, 'climbing' the tree till its root. This observation will play an important role in the definition of a version of the protocol $\mathcal{P}_e$ whose state space size remains *moderate* in all the cases. Firstable, I present the general properties of $\mathcal{P}_e$, as it has been defined.

Proposition 1. *Given a tree Boolean expression e, the protocol $\mathcal{P}^e$ in Definition 3 : a) Is correct, computing the value of e applied to the values computed by the protocols $\mathcal{P}_j$; b) For $S = \sum_{j=1}^{k} |Q_j|$ we have $|Q^e| \leq S\, 2^h$, where $h =$ height(e) ; c) For all $c > 1$, whenever e is c-balanced (which means height$(e) \leq c \log k$) we have $|Q^e| \leq k\, S\, 2^c$; d) In the (very) worst case we have $|Q^e| \leq S\, 2^k$.*

Proof. a) It is an immediate consequence of Theorem 1, since $\mathcal{P}^e$ is obtained by iterative application of Definition 3. b) Obvious, since $|Q^e| = \sum_{j=1}^{k} |Q_j| \times 2^{d_j}$. c) and d) are immediate corollaries of b). □

In this way, $|Q^e|$ will have a moderate size and preserves succinctness of the protocol arguments *in the average case*, since it is well known that balanced trees are *highly probable*. However, we are interested in an universal succinctness result that will be obtained by reducing the number of bits s added at the definition of the states in $|Q^e|$. This reduction will be proved correct by applying the following

Fact 1 *Any tree of size S (where here size means its number of leaves) contains some leave at depth $\log S$. This result is immediate, by bipartition.*

Then we define the *reduced* protocol $\mathcal{P}^e_{red}$ as follows:

Definition 4. *Given a tree Boolean expression e, we define the protocol $\mathcal{P}^e_{red} = (Q^e_{red}, I^e_{red}, T^e_{red}, O^e_{red})$ taking $Q^e_{red} = \cup_{i=1}^{k} (Q_i^{pt_i} \times \{0,1\}^{0..bd_i})$, where $bd_i = min(\{d_i, \lceil \log k \rceil\})$; $I^e_{red}(x) = \sum_{i=1}^{k} I_i(x)^0$, where we add $bd_i + 1$ 0's to obtain states in Q^e_{red} ; $O^e_{red}((q_i, s)) = s[0]$; and T^e_{red} contains the* extended *transitions*

$$(q^i_{j_1}, q^i_{j_2} \to q^i_{j'_1}, q^i_{j'_2}) \in T_i \implies ((q^i_{j_1}, s_1), (q^i_{j_2}, s_2) \to (q^i_{j'_1}, s_1), (q^i_{j'_2}, s_2)) \in T^e_{red}$$

the communication transitions $(q^i_{j_1}, s_1), (q^{i'}_{j_2}, s_2) \to (q^i_{j_1}, s'_1), (q^{i'}_{j_2}, s'_2)$ *when* $d_i \leq \lceil \log k \rceil$, *with* $s'_1[0] = s'_1[1]$, $s'_2[0] = s'_1[1]$ *and* $s'_1[j] = s_1[j]$, $s'_2[j] = s_2[j]$ *for* $j > 0$; *and* $(q^i_{j_1}, s_1), (q^{i'}_{j_2}, s_2) \to (q^i_{j_1}, s'_1), (q^{i'}_{j_2}, s'_2)$, *now only for* $i \neq i'$ *such that* $(d_i\text{–}lcp_{i,i'} < \lceil \log k \rceil) \wedge (d'_i\text{–}lcp_{i,i'} < \lceil \log k \rceil) \wedge ((d_i\text{–}lcp_{i,i'} < \lceil \log k \rceil - 1) \vee (d'_i\text{–}lcp_{i,i'} < \lceil \log k \rceil - 1))$, *and in such a case s'_1 and s'_2 are obtained from s_1 and s_2 by turning the $(lcp_{i,i'} - d_i + \lceil \log k \rceil + 1)$-th bit of s_1 and the $(lcp_{i,i'} - d_{i'} + \lceil \log k \rceil + 1)$-th bit of s_2 into $v_1 \odot_{pt_{i,i'}} v_2$, where $v_1 = s_1[lcp_{i,i'} - d_i + \lceil \log k \rceil + 2]$ when $lcp_{i,i'} + 1 \neq d_i$, and $v_1 = O_1(q^i_{j_1})$, otherwise; and analogously $v_2 = s_2[lcp_{i,i'} - d'_i + \lceil \log k \rceil + 2]$, when $lcp_{i,i'} + 1 \neq d'_i$, and $v_2 = O_2(q^{i'}_{j_2})$, otherwise. Where in order to simplify the reading I have assumed $bd_i < d_i$ and $bd'_i < d'_i$, since in the opposite case the full sequence of bits added in Definition 4 is preserved, and then we work directly with the $(lcp_{i,i'} + 1)$-th bit, as done there.*

I hope that the cumbersome notation above will not hide the 'simple' pruning mechanism of the added sequences, by preserving at most the $\lceil log\, k \rceil$ last bits attached at the agents of each protocol. These ones keep the values of the subexpressions that correspond to their closest ancestors. Now each agent will only be involved on those computations, leaving the ones for its oldest ancestors to the agents of the protocols that are in any of their first $\lceil log\, k \rceil$ generations of successors. Besides, all the agents (q, s) contain one bit $(s[0])$ for the value of the full expression e. Their values are obtained by firing the first communication transitions in T^e_{red}. This is enough, since Fact 1 guarantees that the correct computation of any subexpression of e is still possible. This can be proved by induction, proceeding *bottom up* at the tree, because at both sides of each internal node there will be at least one agent (by the way, all those computing any of its closest protocols) that has correctly computed the value of the corresponding descendant. These two agents can communicate, since at least one of them still disposes of one additional bit for the saving of the computed value, thus accomplishing the correct computation of the value of the corresponding subexpression.

Proposition 2. *Given a tree Boolean expression e, the protocol $\mathcal{P}^e_{red}$ defined in Definition 4 : a) Is correct, computing the value of e applied to the values computed by the protocols $\mathcal{P}_j$; b) For $S = \sum_{j=1}^{k} |Q_j|$ we have $|Q^e| \leq k\, S$; c) Whenever the subprotocols $\mathcal{P}_j$ are silent, $\mathcal{P}^e_{red}$ is too.*

Proof. a) (sketch) We can prove, by induction on the height of each subexpression e_s of e that all the agents in the set $\mathfrak{P}_s$ of arguments of e_s will get a (partial) consensus that includes the value of each of them at the supplied input, and those of each of the subexpressions $e_{s'}$ of e_s, whose values will remain stable at any of the added bits to keep those values. Moreover, for $k_s = |\mathfrak{P}_s|$, by applying Fact 1 some of the protocols $\mathcal{P}_{i_s} \in \mathfrak{P}_s$ is at most $\lceil log\, k_s \rceil$ levels below the root of e_s, and since $k_s < k$ all the agents working on that protocol contain a bit for the computation of the value of e_s. Then they will meet some of the agents including a bit for the computation of the value of the other child of the root of e_s, and applying $\odot_s$ the correct value of e_s will be transferred to the added bit for its computation at the agents in Q_{i_s}. In particular, the agents with $d_i \leq \lceil log\, k \rceil$ will compute the value of e, and they communicate it to all the agents working on $\mathcal{P}^e_{red}$.

b) Obvious, since we added at most $\lceil log\, k \rceil$ bits to each state of the protocols $\mathcal{P}_i$. c) We can extend the inductive proof above including the silenceless requirement. Once the agents working at each protocol argument stabilize, the communication transitions will compute the final values of all the added bits, so that no later transition will change them anymore. □

One of the facts proved in [9] to obtain their main theorem stating that it is possible to obtain succinct protocols for the computation of Presburger predicates (this means that they have a polynomial number of states in the size of the predicate, when it is presented by means of a *Boolean combination* e of *threshold* and *modulo* predicates) is that the class of predicates that are

definable by succinct protocols is closed under Boolean operations. Their quite involved proof needs to combine a great number of complex techniques, and those needed to prove the latter result are particularly involved and introduce reverse transitions that make the obtained protocol no silent. Instead, using distributed protocols I have proved Proposition 2 above that in particular will give us succinct protocols for all the Presburger predicates as soon as we have succinct protocols to compute *threshold* and *modulo* predicates.

In the following sections we will see that the distributed approach also provides simple ways to reduce the number of states of the protocol that implements a certain predicate. These are obtained taking profit of the binary representation of natural numbers, as has been done in [10, 9] .

4 Counting with small agents

In [10] a succinct protocol for counting populations based on the use of states for accumulating 2^k units is presented. It transfers to the framework of population protocols the *binary representation* of integer numbers, although only internally at the protocol, while the input is still presented in *unary* form. To be precise, the considered problem was to decide if the size of the input, which means the number of agents in it, is bigger or equal than a certain natural number n.

Their protocol has $Q = \{2^k \mid k \in \mathbb{N},\ k \leq log\, n\} \cup \{0, n\}$ and uses the fact that $2^k + 2^k = 2^{k+1}$ to promote the agents to higher powers. In order to detect the bound value n they introduce a multiway transition that converts the collection of agents representing the bits of n into one agent n (e.g. for $n = 45$, it turns 32+8+4+1 into 45+0+0+0). State n is the only one with output 1, and once it is reached it turns all the other agents into n by means of communicating transitions. Besides, in order to avoid that along an execution the value n will be exceeded (e.g. turning 32+32 into 64+0) without reaching it exactly, they need also *reverse* transitions that undo the promotion to higher powers. And in order to get a classical protocol, they have to encode the multiway transition generating the state n into a collection of 2-way transitions, something that requires again the introduction of reverse transitions.

Next, I present a pure protocol for the counting task, that later I will present as a distributed one, which will be easily modified to solve another more general counting problems. I will show that the introduction of a few more states (doubling at most its number) makes possible a very efficient protocol that does not include any reverse transition.

Definition 5. *For each bound $n > 0$ we define the pure protocol $\mathcal{P}_n^{count} = (Q_n, I_n, T_n, O_n)$ with $Q_n = Q_n^{pow} \cup \{0\} \cup Q_n^{apr}$, where*

$$Q_n^{pow} = \{\, 2^k \mid k \in 0..(l+1) \,\} \qquad Q_n^{apr} = \{q_n^k \mid 2 \leq k \leq p \,\} \ \text{with}\ q_n^k = 2^l + \sum_{j=2}^{k} 2^{i_j}$$

where $l = \lfloor log\ n \rfloor + 1$, the 2-base digits of n are $d_l \dots d_0$ ($n = d_j \dots d_0$) and $1_n = \{i \mid 0 \leq i \leq l \wedge d_i = 1\}$ taking $1_n = \{i_1, \dots i_p\}$ with $i_1 = l$ and $i_j > i_{j+1}$ for $j \in 1..(p-1)$, the decreasing enumeration of the bits of n.

$I_n = \{2^k \mid k \leq l\}$. $T_n = T_n^{up} \cup T_n^{apx} \cup T_n^{goal} \cup T_n^{comm}$, *where:*

$$\begin{array}{lll} T_n^{up} & : & 2^k, 2^k \to 2^{k+1}, 0 \qquad k \in 0..l \\ T_n^{apx} & : & q_n^k, 2^{i_{k+1}} \to q_n^{k+1}, 0 \quad k \in 1..(p-1) \\ T_n^{goal} & : & q_n^k, 2^j \to n, n \qquad k \in 1..(p-1)\,,\ j > i_{k+1} \\ T_n^{comm} & : & f, q \to f, f \qquad f \in \{2^{l+1}, n\}\,,\ q \notin \{2^{l+1}, n\} \end{array}$$

The example $n = 45$ will illustrate this definition. We have $Q_{45} = \{1, 2, 4, 8, 16, 32, 64, 0, 40, 44, 45\}$ and some of the transitions in T_{45}, one for each of its subsets, are: $16, 16 \to 32$; $40, 4 \to 44, 0$; $40, 8 \to 45, 45$; $64, 1 \to 64, 64$.

Note that, even if we are including in Q_n the states in Q_n^{apr}, we are maintaining its logarithmic size, since $l \in O(\log n)$, so that between 2^l and n we introduce at most l interpolations (in the example, only two: 40 and 44), instead of all the values in the interval (33..45 in the example.)

Proposition 3. *For each initial configuration $C_0 \in \mathbb{N}^{I_n}$, $C_0 = k_0 \cdot 2^0, \ldots, k_l \cdot 2^l$, $\mathcal{P}_n^{count}$ decides whether $\sum_{j=0}^{l} k_j\, 2^j \geq n$. Besides, $|Q_n| \leq 2\, l \leq 2\, \log n$.*

Proof. We decompose the set of transitions into $T_n^{small} = T_n^{up} \cup T_n^{apx}$, and $T_n^{large} = T_n^{goal} \cup T_n^{comm}$. T_n^{small} contains the transitions that preserve the sum of the values of the evolving agents, while T_n^{large} includes those 'adding' two agents whose sum is greater or equal than n, and producing two such values. When $\sum_{j=0} k_j\, 2^j < n$, it is obvious that we can never apply a transition in T_n^{large}, so that for any reachable configuration C we have $O(C) = 0$, since $2^{l+1}, n \notin C$. Instead, when $\sum_{j=0} k_j^l\, 2^l \geq n$, the application of each transition in T_n will produce two agents with a total value greater or equal than the sum of the two evolving ones, till we arrive to an stable configuration C. Whenever it contains any *final* state (2^{l+1} or n), by applying the communication transitions we reach a consensus configuration C', with $O(C') = 1$. An inductive argument proves that eventually we reach such a configuration, based on the fact that $2^g > \sum_{k=0}^{g-1} 2^k$, that combined with $\sum_{j=0} k_j\, 2^l \geq n$ implies that either $\{\{2^k, 2^k\}\} \leq C$, for some $k \in 0..l$, or $\{\{q_n^l, 2^{j-1}\}\} \leq C$, with $j \in 0..l$, so that C would not be stable. □

Discussion 1 As explained in the introduction of this section, the presented protocol is an adaptation of the succinct protocol for counting populations in [10]. A fast comparison could conclude that its (correct) behaviour is based on the same 'essential' principle: simply the *binary representation* of integer numbers. However, looking into the details, several important differences arise, all of them (but a single one) representing important advantages of my solution here. I will start with the only (minor) disadvantage: here I have $|Q_n| \leq 2\, \log n$ states, while the protocol in [10] had at most $\log n + 2$. Anyway, I maintain $|Q_n| \in O(\log n)$.

Next the main advantage, from which emanate several interesting properties. $\mathcal{P}_n^{count}$ satisfies **total termination**: by the way, revising the correctness proof above it is easy to quantify the duration of its two phases, seeing that there are at most n transitions in T_n^{small}, and at most n in T_n^{large}. After executing them we reach a silent configuration. So we have an extremely fast *silent* protocol that besides **always terminates** (once silent transitions are not considered).

This is a consequence of the fact that I have 'totally' avoided both *multiway transitions* and also *reverse* transitions by themselves, such as $32, 0 \to 16, 16$. Obviously, any introduced reverse transition implies **no** total termination, and if not 'explicitly' somehow avoided, also **no silence** when a consensus is reached. This usually causes a very poor efficiency, as it is the case in [10]. For instance, in order to reach any (pre)successful configuration (those including one n agent) they need to reach just before the configuration $2 \cdot 2^{l-1} + (2^l - 2) \cdot 0$. Then the probability to execute the transition $2^{l-1}, 2^{l-1} \to 2^l, 0$ is $\frac{2}{n(n-1}$).

But these are far from being the worst news. The real problem is that, as a consequence, it is very probable that after executing the following two transitions the reached configuration will be $4 \cdot 2^{l-2} + (2^l - 4) \cdot 0$ and then to recover our initial transition above we need two hit events whose probability is fairly smaller (for the same reason!) than $\frac{2}{(n/2)(n/2-1)^2}$. In fact, together with another author, the authors of [10] have presented in [11] a very nice *executable* paper where the reader can find a painful simulation of the protocol (since confirming my predictions above, she must be very very lucky to see its termination, even for very small values of n.)

In fact, these 'nested' reverse mechanisms are used by the developers of *sequential protocols* when they want to control that the second phase of a protocol can start after obtaining (with a certain 'high' probability) the result of the first phase (see e.g. [3]). For that they need to measure the passage of an (expected) exponential period of time, and this can be done by checking the termination of a counting protocol somehow 'similar' (but simpler) to the one in [10].

In Section 2 we have seen how we should introduce the agents corresponding to the input variables that do not appear in a certain predicate when we want to put together two protocols, for instance for the computation of the conjunction $\overline{\phi}(x_1, ..., x_k, y_1, ..., y_k) = \phi_1(x_1, ..., x_k) \wedge \phi_2(y_1, ..., y_k)$. A quite natural way of introducing the required dummy agents in a protocol that is making some arithmetic calculation (at the end anyone is doing that, but at the definition of the protocol this can be 'hidden'), as it is the case of the succinct counting protocols, is to introduce them as 0 agents. Obviously, these protocols will (qualitatively) remain correct after such an addition. However, any *superfluous* 0 agent will dramatically delay (even more!) the stabilization of the protocol, since they *trigger* the execution of the reverse transitions undoing the work of the transitions that produce the progress toward the bound n. Therefore, we should avoid this undesired effect introducing the dummy agents as totally 'external' to the working of the protocol, only participating in the dissemination of its output.

Finally, protocols including any reverse **cannot** be WS^3 protocols, that are the protocols that again the authors of [10], together with another forth author, have shown to be *efficiently verifiable* in [12]. Instead, with a simple modification in order to satisfy the second condition in the definition of WS^3 protocols, my protocol $\mathcal{P}_n^{count}$ becomes WS^3, and thus efficiently verifiable.

From the pure protocols $\mathcal{P}_n^{count}$ we can derive a distributed protocol for the addition of $(l+1)$-byte numbers in the set $B_l = 0..(2^{l+1} - 1)$. For each $v \in B_l$ we define the set 1_v as 1_n in Definition 5, and we take $D(v) = \{\{\, 2^k \mid k \in 1_v \,\}\}$.

Definition 6. *For each bound $n \in \mathbb{N}$ the distributed protocol $\mathcal{P}_n^{bad}$ for the addition of $(l+1)$-byte numbers is defined as $\mathcal{P}_n^{bad} = (Q_n, I^{bad}, T_n, O_n)$, with Q_n, T_n and O_n as in Definition 5, $X = B_l$, and $I^{bad} : B_l \to Q_n$, with $I^{bad}(b) = D(b)$.*

Corollary 1. *For each bound $n \in \mathbb{N}$ the distributed protocol $\mathcal{P}_n^{bad}$ decides if the given initial population $\alpha \in \mathbb{N}^{B_l}$ satisfies $\sum_{b\in\alpha} b \geq n$.*

Finally, I present a distributed protocol for the computation of *linear sums* $a_1\, x_1 + \cdots + a_m\, x_m$, with $a_i, x_i \in \mathbb{N}$, for unary representation of the input $\overline{x}$. In a similar way we can obtain the distributed protocol for binary inputs.

Definition 7. *For each vector $\overline{a} \in \mathbb{N}^m$ and any bound $n \in \mathbb{N}$, the distributed protocol $\mathcal{P}_n^{\overline{a}}$ for the computation of linear sums $\sum_{i=1}^m a_i\, x_i$ is defined as $\mathcal{P}_n^{\overline{a}} = (Q_n, I_n^{\overline{a}}, T_n, O_n)$, with Q_n, T_n and O_n as in Definition 5, and $I_n^{\overline{a}} : X \to Q_n$, with $X = \{x_1, \ldots, x_m\}$, and $I_n^{\overline{a}}(x_i) = D'(a_i)$, where $D'(a)$ is defined as $D(a)$ for $a \in B_l$, and taking $D(a) = \{\{2^{l+1}\}\}$, for $a \geq 2^{l+1}$.*

Corollary 2. *For each vector $\overline{a} \in \mathbb{N}^m$, and any bound $n \in \mathbb{N}$, the distributed protocol $\mathcal{P}_n^{\overline{a}}$ decides whether $\sum_{i=1}^m a_i \times x_i \geq n$.*

5 Distributed protocols for Presburger predicates

It is well known that classical and pure population protocols compute exactly the family of *Presburger predicates*. Here we recall its modular characterization as the Boolean closure of the *threshold* and *remainder* predicates. Both are defined based on linear expressions $\sum_{i=1}^s a_i\, x_i - \sum_{j=1}^g b_i\, y_i$, with $a_i, b_j > 0$. In the first case we check if the sum is greater or equal than some $n \in \mathbb{Z}$, while in the second we check if it is equal to some value $n \in 0..(m-1)$, modulo some given $m > 0$.

It is not difficult to see that instead of the full family of *threshold* and *remainder* predicates, we can take as basis for the generation of Presburger predicates the family of *positive thresholds*, that correspond to positive bounds $n > 0$, and the *bounded positive remainder* predicates, that only include positive coefficients a_i, and check if the considered sum is greater or equal to n, instead of equality.

Next, I present a succinct distributed protocol for the computation of positive threshold predicates, that reduces the number of needed states, as in Section 4. Bounded positive remainder predicates can be implemented following the same ideas, although in that case the role of the transitions $2^k - 2^k = 0$ simplifying the configurations, must be played by the transitions $km + x \equiv_m x$.

Succinct distributed protocols for positive threshold predicates

I tried to develop these protocols following the same ideas that for $\mathcal{P}_n^{\overline{a}}$ in Definition 7, but in the beginning this seemed not possible. Why? Because the (simpler) predicate computed there was *true-monotonic:* whenever an input population produces an affirmative output, any bigger population also leads to 1. This is exploited in the design and in the correctness proof of that protocol. Instead, the negative part of the linear expression now is clearly 'anti-monotonic'.

We need the *balancing* transitions that use $2^k+(-2^k)=0$ to take into account the contribution of negative agents. If we try to manage *approximation* agents as in Definition 7, sometimes we will need to turn back these *approximations* in order to recover the required 'small' agents that will make possible some balancing transition. But in order to undo these transitions we need 0-agent partners. It could be the case that we have run out of them, and then we would be stuck.

But going back to a protocol that only uses numerical agents with values 2^k, as developed in [9], it is possible to circle these difficulties. However, now to check our goal we will need to detect the simultaneous presence of the agents in $D(n)$. This is done in [9] by means of a multiway transition. Instead, we find here another application of our distributed protocols, by choosing an input function that generates a sufficient, but not too large, quantity of agents to play all the roles that are needed to get the desired protocol.

Definition 8. *For each pair of vectors $\overline{a} \in \mathbb{N}^s, \overline{b} \in \mathbb{N}^g$, and bound $n > 0$, the distributed protocol $\mathcal{P}^{thr}_{\overline{a},\overline{b},n}$ for deciding the positive threshold predicate $\phi^{thr}_{\overline{a},\overline{b},n} \equiv (\sum_{i=1}^{s} a_i\, x_i - \sum_{j=1}^{g} b_j\, y_j \;\geq\; n)$ is defined as $\mathcal{P}^{thr}_{\overline{a},\overline{b},n} = (Q^{thr}, I^{thr}, T^{thr}, O^{thr})$, where taking $z = max(\{a_i \mid i \in 1..s\} \cup \{\, b_j \mid j \in 1..g \,\} \cup \{n\})$ and $vl = \lfloor log\, z \rfloor + 1$, and being $d_h \ldots d_0$ the 2-base digits of n, and 1_n as in Definition 5, we have:*

$Q^{thr} = \{(2^k, b) \mid k \leq h,\, b \in \{0,1\}\} \cup \{(2^k, 1) \mid h < k \leq l\} \cup \{-2^k \mid k \leq l\} \cup \{(0,0),(0,1)\} \cup Q^{thr}_{lead,n}$, *where* $Q^{thr}_{lead,n} = Qb_n \cup Qe_n$
with $Qb_n = \{(i, ch, o) \mid i \in 1_n,\, ch, o \in \{0,1\}\}$ *and* $Qe_n = \{el_i \mid i \in 1_n\}$

In this case, the set $X \cup Y$ will be our input alphabet, with $X = \{x_1, \ldots, x_s\}$ and $Y = \{y_1, \ldots, y_g\}$, and we take

$$I^{thr}(x_i) = \{\{(2^i, 0) \mid i \in 1_{a_i}\}\} + (\lceil log\, a_i \rceil - |1_{a_i}|) \cdot (0,0) + Qe_n$$
$$I^{thr}(y_j) = \{\{-2^j \mid j \in 1_{b_j}\}\} + (\lceil log\, b_j \rceil - |1_{b_j}|) \cdot (0,0)$$

where we define the sets 1_{a_i} and 1_{b_j} in an analogous way as we defined 1_n.

$O(q,b) = b$, *for* $q \geq n$; $O(q) = 0$, *for* $q < 0$; $O((i,ch,o)) = o$; $O(el_i) = 0$.

$T^{thr} = T^{thr}_{up} \cup T^{thr}_{down} \cup T^{thr}_{cancel} \cup T^{thr}_{move} \cup T^{thr}_{lead} \cup T^{thr}_{comm} \cup T^{thr}_{check}$.

T^{thr}_{up} : $(2^k, b_1), (2^k, b_2) \to (2^{k+1}, b_1), (0, b_2)$ $\quad k \in 0..(l-1)$
: $(-2^k, b_1), (-2^k, b_2) \to (-2^{k+1}, b_1), (0, b_2)$ $\quad k \in 0..(l-1)$

T^{thr}_{down} : $reverse(T^{thr}_{up})$

T^{thr}_{cancel} : $(2^k, b), -2^k \to (0,b), (0,b)$ $\quad k \in 0..l$
: $(k, ch, b), -2^k \to (0,b), (0,b)$ $\quad k \in 0..l$

T^{thr}_{move} : $(2^k, b), el_k \to (k, 0, b), (0, b)$ $\quad k \in 1_n$

T^{thr}_{lead} : $l_1, (k, ch, b) \to l_1, (2^k, b)$ $\quad l_1 \in \{\, (k, ch, b), el_k \mid ch, b \in \{0,1\}\,\}$ $\quad k \in 0..l$
: $l_1, el_k \to l_1, (0,0)$ $\quad l_1 \in \{\, (k, ch, b), el_k \mid ch, b \in \{0,1\}\,\}$ $\quad k \in 0..l$

T^{thr}_{comm} *propagates any output, either positive or negative, anywhere is possible.*

Finally, to check that all the leaders are busy (detecting a set of positive agents whose sum is n), we have the transitions in T^{thr}_{test}. We consider the enumeration in decreasing order of $1_n = \{i_1, \ldots, i_p\}$ (e.g. $1_{45} = \{5, 3, 2, 0\}$), and we have:

$$(i_1, v, b_1), (i_2, 0, b_2) \to (i_1, v, b_1), (i_2, 1, b_2)$$
$$(i_k, 1, b_1), (i_{k+1}, 0, b_2) \to (i_k, 0, b_1), (i_{k+1}, 1, b_2) \quad 1 < k < (p-1)$$
$$(i_{p-1}, 1, b_1), (i_p, 0, b_2) \to (i_{p-1}, 0, b_1), (i_p, 1, 1)$$

This corresponds to the general case of the definition: $p>2$ *. When* $p=1$ *, which corresponds to* $n=2^h$ *, we can simply put the value* 2^h *in the set of states with fix output* 1 *, and then we can avoid the leader mechanism and the* test *transitions. While when* $p=2$ *, we simply take* $(i_1,v,b_1),(i_2,0,b_2)\to(i_1,v,b_1),(i_2,1,1)$ *.*

Let me start by clarifying that even if I talk about *leaders*, they are not such in the usual meaning: they are ordinary agents created by each unit of some of the elements in the input alphabet. Next we will see that the protocol includes a mechanism for the selection of *unique leaders* between the initial population of leader agents, as in [15, 3]. This is why I keep this appellative in the definition.

The first component of the protocol is the *balancing* procedure that will compensate equal agents of different sign, by means of the transitions in T^{thr}_{cancel}. In order to get the matching pairs we have the transitions in T^{thr}_{down}, that could require the presence of 0-agents. This is why we introduce those agents in the definition of I^{thr}. They will be (more than) enough to do their task.

Fairness will guarantee that eventually we will have either no negative or no positive agents. In the latter case, a consensus on 0 output will be reached, by applying the transitions in T^{thr}_{comm}. In the former, the protocol must check whether the remaining positive units are n or more. The idea (already used in [9]) is that once only non negative agents remain, by means of the transitions in T^{thr}_{down} and T^{thr}_{up} we can distribute the remaining units in such a way that we can reach a configuration C', including a set of agents corresponding to the elements of 1_n, if and only if $\phi^{thr}_{\overline{a},\overline{b},n}(\overline{x},\overline{y})=1$.

We need a procedure to detect this situation. We do it by means of the transitions in T^{thr}_{test}, assuming that previously the transitions in T^{thr}_{lead} have selected a family of *unique leaders*, one for each value in 1_n. It will be enough to check that all these leaders are busy to know that the current configuration contains positive agents whose total value is greater or equal than n. This works thanks to the transitions in T^{thr}_{move}, that after collecting 2^k positive units of the configuration in one 2^k-agent, transfer them to the corresponding leader agent when it is empty, turning it into busy. Any successful test will 'mark' the output field of the i_p-leader agent turning it into $(i_p,1,1)$. Next, this 1-output will be broadcasted using the transitions in T^{thr}_{comm}.

Certainly, when we check the leaders 'too early' we could get a *hurried* positive, that however will not cause any problem, since whenever the balancing procedure discussed above will terminate, either we have no positive agent, and then we reach a stable 0 consensus by executing the transitions in T^{thr}_{comm}; or no negative agent remains in the configuration. When this is the case we can (and we will) fill all the leader agents at the same time if and only if the value of the configuration is greater or equal than n. In the positive case, any test after all the leader agents are busy will succeed, and then the produced 1-output will reach everywhere and the obtained consensus will remain stable. While in the negative case, no test can succeed since the busy leader agents 'crossed' by the test will remain busy forever, since no negative agent remains to empty them. As a consequence, any successful test now implies that all the leader agents are

busy at the same time, and therefore the value of the configuration must be greater or equal than n, against our current hypothesis.

Note that in the last reasoning above it is crucial that there is no negative agent around, since as long as they are present they could 'inadvertently' cause that a test could succeed, even when the total value of the positive agents at the configuration is less than n. This malfunction could occur because some positive units already checked by a test could be 'liberated' by a negative agent that expects to remove them by executing a later cancellation transition, but instead those liberated positive agents could fill any of the leader agents still to check, thus cheating the testing procedure since the same units would be used twice. But fortunately, this cause no problem in our protocol, as explained above.

If we compare the solution with that in [9], we can see that I do not need any lock for checking that n positive units have been collected. In [9] success is claimed by the firing of the multiway transition, once and again, but this was only possible because they use a special kind of (standard!) leaders, that they call *voters*. Once more, my proposal here only uses regular agents, thanks to the flexibility of distributed protocols, also avoiding the centralization imposed by those voters.

Perhaps you could think that using multiway transitions (that later could be turned into regular ones) I could avoid the introduction of my leader agents. This is not possible: if we look for the agents corresponding to 1_n anywhere at the configuration, we could succeed in several directions. But it would be impossible to avoid that any agent that has been affirmatively tested, but then is immediately converted into one $(0,0)$ agent by one cancellation transition, could be next turned into $(0,1)$ by a communication with any 1-output agent. But if this happens just when the last negative agent has been cancelled, turning the value of the configuration below n, the broadcasted 1 could not be corrected anymore, thus producing a wrong output. This cannot happen when the removed positive agent is a busy leader, since empty leaders always claim 0 as output.

Certainly, as brightly discussed in [18], leaders are always required to generate any 'asymmetric' behaviour of the agents in a protocol, and they always introduce a certain *coordination*, which requires some centralization, but that implied by the leaders in this protocol seems not big, since they are only involved in a small part of the transitions of the protocol.

Let me next just to present the theorem that states the correctness of $\mathcal{P}^{thr}_{\overline{a},\overline{b},n}$. Its formal proof would just present in a more formal way the informal reasoning above. However, I will include below some technical results that are needed in that formal proof.

Theorem 2. *For each pair of vectors $\overline{a} \in \mathbb{N}^s, \overline{b} \in \mathbb{N}^g$, and any bound $n > 0$, the distributed protocol $\mathcal{P}^{thr}_{\overline{a},\overline{b},n}$ decides the positive threshold predicate $\phi^{thr}_{\overline{a},\overline{b},n} \equiv (\sum_{i=1}^{s} a_i\, x_i - \sum_{j=1}^{g} b_j\, y_j \geq n)$.*

First, we define the *value* of a configuration C of $\mathcal{P}^{thr}_{\overline{a},\overline{b},n}$ as the sum of all its numerical agents, adding 2^i for each busy leader $(i,v,b) \in C$. All the transitions of the protocol preserve the value of the configuration, so that for

any reachable configuration C from any initial configuration $I(\overline{x},\overline{y})$, we have $value(C) = \sum_{i=1}^{s} a_i\, x_i - \sum_{j=1}^{g} b_j\, y_j$.

Definition 9. *We say that a configuration C of $\mathcal{P}^{thr}_{\overline{a},\overline{b},n}$ is* simple positive *(resp. negative), corresponding to $0 \leq v < 2^{l+1}$ (resp. $-(2^{l+1}) < v < 0$) when for each $k \in 1_v$ (resp. 1_{-v}) we have a single agent $(2^k, b) \in C$ and $C((0,t)) + C((0,f)) = |\{k : k<l \wedge k \notin 1_v\}|$ (resp. $\in 1_{-v}$) and $C(q)=0$, for any other $q \in Q^{thr}$.*

This definition captures the fact that along the computations of $\mathcal{P}^{thr}_{\overline{a},\overline{b},n}$ we have enough 0-agents around, since the initial configurations can be decomposed into a disjoint union of simple configurations, by applying the definition of I^{thr}.

Lemma 1. *Any simple positive (resp. negative) configuration C of $\mathcal{P}^{thr}_{\overline{a},\overline{b},n}$ can be transformed, by applying transitions in T^{thr}_{rad}, into another configuration $C' = C'' + \{\{(1,b)\}\}$, (resp. $(-1,b)$) where C'' either only contains agents $(0,b')$, or is also a simple positive (resp. negative) configuration C of the protocol.*

Corollary 3. *From any initial configuration C_0 of $\mathcal{P}^{thr}_{\overline{a},\overline{b},n}$ we can reach C' that contains no (q,b) with $q<0$, or no (q,b) with $q>0$, and $value(C') = value(C)$.*

Proposition 4. *For any initial configuration C_0 of $\mathcal{P}^{thr}_{\overline{a},\overline{b},n}$ with $value(C_0) \geq n$, if we have $C_0 \xrightarrow{*} C'$, we also have $C' \xrightarrow{*} C''$, where C'' contains a* full set of busy leaders*, i.e. a busy leader agent for each k with $d_k = 1$.*

Proof. (Of Theorem 2) Any fair execution will reach a configuration which contains one single k-leader, for each $k \in 1_n$. Then, fairness also guarantees, as stated by Corollary 3, that we will reach a configuration which either contains no negative or no positive agent. Next, Proposition 4 guarantees that we will reach a configuration with single leaders, all of them busy, if and only if $\sum_{i=1}^{s} a_i\, x_i - \sum_{j=1}^{g} b_j\, y_j \geq n$. These situations will be stable. In particular, in the affirmative case the execution of the transitions in T^{thr}_{test} will fix 1 as output of the leaders, and then the transitions in T^{thr}_{comm} will communicate this value to all the other agents. In the opposite case the set of busy leaders will also stabilize and some leader agent will remain empty forever, so that in the following any test will fail when reaching it. Then, the empty leaders will communicate their 0-output using the transitions in T^{thr}_{comm} and the consensus on the output 0 will stabilize. □

6 Concluding Remarks and some Future Work

Only after studying in detail most of the great papers in the References below I got the inspiration to discover the narrow *bottleneck* that one to one relationship between the units of the input and those of the input configuration, is imposing to classical population protocols. My distributed protocols, that in fact are only a particular case of *pure* protocols, are in my opinion a simple and elegant proposal that makes much easier the development of simple and more efficient protocols.

So, the introduction of my distributed protocols does not really mean a generalization of the definition of population protocol, but just the confirmation of the fact that using this suggestive class of pure protocols, we can directly manage *specialized* agents, without (formally) contradicting the uniformity of the (global) *code* that governs the behaviour of all the agents of the protocol. The agents can be now designed in a way that, depending on their initial state, each one can only reach a limited set of states (its type!). Then we can (in practice) specialize the code for each type, simply including in it the transitions that have as first agent one member of the type, although of course the met agent could be any.

Since I did not want to generalize the notion of population protocol, I have maintained the uniform participation of all the agents in the dissemination of the global (final) output. However, you can find in Definitions 3, 4 two clear examples of *structured* protocols, based on a family of *subprotocols* (those corresponding to all the internal nodes of the expression e) 'locally' conceived as ordinary protocols, but whose (local) output must be *internalized*. This is done in a way that only the agents participating in each subprotocol compute the corresponding local output, and it is only transmmited to other agents as far as they will need it to accomplish their tasks . This idea could be transferred to the global output, so that only the 'principal' agents will compute and transmit the final output, while the 'auxiliary' agents only would help doing its part, and even could disappear after accomplishing it, following the old notion of *coroutine* for parallel programming. I plan to continue the exploration of these *structuring* ideas, looking in particular for a *higher level* programming language for population protocols. In this case [16, 2] are interesting starting points.

As continuation of this work, based on [12], I have already shown that distributed protocols will also be easy to prove correct, and once more, even easier than classical protocols, specially when the corresponding proofs will be done by hand, but probably also when done in a mechanical way, if the tools can be adapted to take advantage of the *modularity* of protocols. Besides, I have just submitted [14] where I complete the development of *succinct* distributed protocols for Presburger predicates, as in [9], providing an alternative succinct threshold protocol, since finally I found the way to do it following the ideas in Definition 5, thus obtaining a silent protocol totally free of reverse transitions. And a more involved application of these ideas also produced the required succinct *remainder* protocol.

Acknowledgement. I sincerely thank Javier Esparza, that introduced me to Population Protocols, during my stay at TU Munich in April 2017. I also appreciate the indications of the referees of a previous version of this paper. Following them the presentation has been improved, making the paper more readable. This research was partially supported by project PID2019-108528RB-C22 and by Comunidad de Madrid program S2018/TCS-4339 (BLOQUES-CM) co-funded by EIE Funds of the European Union.

References

1. Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-space trade-offs in population protocols. In *28th ACM-SIAM SODA*, pages 2560–2579. SIAM, 2017.
2. Dan Alistarh and Rati Gelashvili. Recent algorithmic advances in population protocols. *SIGACT News*, 49(3):63–73, 2018.
3. Talley Amir, James Aspnes, David Doty, Mahsa Eftekhari, and Eric E. Severson. Message complexity of population protocols. In *34th DISC*, volume 179 of *LIPIcs*, pages 6:1–6:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
4. Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *23rd ACM PODC*, pages 290–299. ACM, 2004.
5. Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Comput.*, 18(4):235–253, 2006.
6. Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *25th ACM PODC*, pages 292–299. ACM, 2006.
7. Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Comput.*, 21(3):183–199, 2008.
8. Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Comput.*, 20(4):279–304, 2007.
9. Michael Blondin, Javier Esparza, Blaise Genest, Martin Helfrich, and Stefan Jaax. Succinct population protocols for presburger arithmetic. In *37th STACS*, volume 154 of *LIPIcs*, pages 40:1–40:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
10. Michael Blondin, Javier Esparza, and Stefan Jaax. Large flocks of small birds: on the minimal size of population protocols. In *35th STACS*, volume 96 of *LIPIcs*, pages 16:1–16:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
11. Michael Blondin, Javier Esparza, Stefan Jaax, and Antonín Kucera. Black ninjas in the dark: Formal analysis of population protocols. In *33rd ACM/IEEE LICS*, pages 1–10. ACM, 2018.
12. Michael Blondin, Javier Esparza, Stefan Jaax, and Philipp J. Meyer. Towards efficient verification of population protocols. In *36th ACM PODC*, pages 423–430. ACM, 2017.
13. Rachel Cummings, David Doty, and David Soloveichik. Probability 1 computation with chemical reaction networks. *Nat. Comput.*, 15(2):245–261, 2016.
14. David de Frutos Escrig. Succinct and fast distributed population protocols. *submitted*, 2021.
15. David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. In *29th DISC*, volume 9363 of *LNCS*, pages 602–616. Springer, 2015.
16. Bartlomiej Dudek and Adrian Kosowski. Universal protocols for information dissemination using emergent signals. In *50th ACM SIGACT STOC*, pages 87–99. ACM, 2018.
17. Rachid Guerraoui and Eric Ruppert. Names trump malice: Tiny mobile agents can tolerate byzantine failures. In *36th ICALP Proc. Part II*, volume 5556 of *LNCS*, pages 484–495. Springer, 2009.
18. Othon Michail and Paul G. Spirakis. Elements of the theory of dynamic networks. *Commun. ACM*, 61(2):72, 2018.