Guidelines for Verification of Population Protocols

Julien Clément, Carole Delporte-Gallet, Hugues Fauconnier, Mihaela Sighireanu

Guidelines for Verification of Population Protocols

Julien Clement Carole Delporte-Gallet Hugues Fauconnier Mihaela Sighireanu
LIAFA
University of Paris Diderot and CNRS
Paris, France
{*jclement,cd,hf,sighirea*}*@liafa.jussieu.fr*

***Abstract*—We address the problem of verification by model-checking of basic population protocol (PP) model of Angluin et al. This problem has received special attention the last two years and new tools have been proposed to deal with. We show that the problem can be solved using the existing model-checking tools, e.g., Spin and Prism. For this, we apply the counting abstraction to obtain an abstract model of a PP which can be efficiently verified by the existing model-checking tools. Moreover, this abstraction is preserves the correct stabilization property of PP. To deal with the fairness assumed in the PP model, we provide two recipes. The first one gives sufficient conditions under which the PP fairness can be replaced by the weak fairness implemented in Spin. We show that this recipe can be applied to several PP models. In the second recipe, we show how to use the probabilistic model-checking and the tool Prism to deal completely with the PP fairness. The correctness of this recipe is based on existing theorems on finite discrete Markov chains.**

***Keywords*-population protocols; Petri Nets; vector addition systems; model-checking; LTL; Spin; Prism**

I. INTRODUCTION

Population protocols [1] are a formal model for sensor networks. Since such networks should become a part of the every day life, it is important to ensure their correctness. This task is now possible in a fully automatic way due to the important progress of verification techniques. Proposed in the early eighties, the model-checking technique [2], [3] allows to check complex correctness properties for models with finite number of states. The system to be verified is formalized in some high level formal language, e.g., CSP or Petri Nets, to obtain a model. The requirements of the system are specified using some logical language, e.g. LTL [4], to obtain a property. Model-checking techniques explore exhaustively the model in order to check that each state of this model satisfies the given property (a reference for model-checking techniques is [5]). Very performant model-checkers, e.g., Spin [6], are now available. They are able to deal with systems of more than 10^{20} states. Moreover, the model-checking techniques have been extended to deal with more complex finite models like the probabilistic ones, e.g., in the Prism [7] model-checker.

Unfortunately, some models of realistic systems have a number of states which exceeds the capacity of finite model-checkers. Thus, the research have focalized on designing abstraction techniques to reduce the model-checking problem of huge systems to the model-checking of smaller systems. One of the first abstraction technique proposed is the counting abstraction [8]. It reduces systems with many identical processes running in parallel to a system which keeps track only of the number of processes which are in some particular state. This abstraction has shown its efficiency for the model-checking of safety properties in cache coherence protocols.

In this work, we apply finite model-checking tools and the counting abstraction technique to verify population protocols.

The model of population protocols (PP) [1] involves individual agents with a very simple behavior, which can be seen as a finite state machine. An important property is the uniformity of the protocol, i.e., the fact that the protocol description is independent on the number of agents (called also the population size) or their identity. When two agents come into range of each other ("meet"), they can exchange information. The agents are anonymous and move in an asynchronous way. At the beginning of the protocol, each agent receives a piece of input. The goal of the protocol is to stabilize each agent into a state in which it outputs the value to be computed by the protocol. Thus, the specification of a population protocol includes the "correct stabilization" (CS) property which requires that each agent stabilizes its computation to an output value which corresponds to a function on the input assigned to agents.

Our contribution is to highlight the use of the existing model-checking techniques to check the correct stabilization property for protocols whose sizes is a fixed constant. For this, we use the counting abstraction to reduce the PP model to a vector addition system model, which is a model computationally equivalent to classical place/transition Petri Nets. We show in Section V that this abstraction allows the verification of the correct stabilization property in a more efficient manner. We also give sufficient conditions to check the CS property under a fairness constraint weaker than the one required in [1]. Based on these theoretical results, we show in Section VI how to perform verification using the Spin tool. Spin is able to verify our benchmark of PP for population size greater than 10^3. The second recipe is presented in Section VII. We highlight that the fairness of the PP model can be exactly captured for the CS property in a probabilistic model where probabilities strictly greater than some $\varepsilon > 0$ are assigned to transitions. Then, we show that the Prism tool is very efficient in the model-checking this property on the abstract model of PP. In addition, we provide a recipe to compute, using Prism, the average number of interactions before the convergence for a PP model. Section IX concludes our work and gives some directions of future research.

Related work: The verification of systems with a huge or unbounded number of processes using the counting abstraction has been introduced in [8]. The PP enter in the class of systems considered by this work. However, the fairness assumption they consider is not the same. We use their abstraction technique to obtain a model on which the finite model-checking tools can be applied. Although we fix the number of processes, we deal with a more complex fairness condition.

In several papers, Pang et al. [9], [10], [11] consider the verification by model-checking of leader election PP on complete graph or ring networks. For complete graphs topologies, they apply directly Spin on the PP model, and thus they obtain results for small population sizes. For the leader election protocol in ring networks, they show that a special fairness is necessary to obtain the correct stabilization. Thus, Spin tool returns false negative for this protocol because it implements a weaker fairness. In, [10], the authors apply Prism to verify the probabilistic version of leader election on rings. These works lead to the development of a new model-checker, called PAT [12]. PAT does model-checking under different fairness conditions including the fairness needed by the leader election PP on rings. However, for the weak fairness, PAT is less efficient than Spin, and for the other fairness conditions, it is able to verify leader election PP with size smaller than 8. In this work, we consider only PP on fully connected graphs, and we manage to verify them for larger populations. Also, we prove theorems on counting abstraction for PP that can be used by any model-checker for the verification of PP models.

[13] shows that the algorithmic verification of finite instances of PP is NP-hard. It provides an incomplete, polynomial algorithm for checking the correctness of the computation for PP whose size is under some fixed constant. We consider the same problem and we prove some theorems in order to use the off the shelf tools for model-checking. With these tools, we obtain better experimental results than those reported by [13].

II. Labeled Transition Systems

At the semantic level, the population protocol model and the verification models we consider are *finite* labeled transition systems.

Definition 1: A finite **labeled transition system** (LTS) is a tuple $\mathcal{T} = \langle S, I, \Sigma, \Delta \rangle$ where S is a finite set of states, $I \subseteq S$ is the set of initial states, Σ is a finite set of labels, and $\Delta \subseteq S \times \Sigma \times S$ is a finite set of labeled transitions.

We also use the notations $s \xrightarrow{\ell} s'$ for $(s, \ell, s') \in \Delta$. The notation $s \rightarrow s'$ stands for $\exists \ell \in \Sigma$ such that $(s, \ell, s') \in \Delta$. We say that a label ℓ is *enabled* in a state s iff there exists s' such that $s \xrightarrow{\ell} s'$. This model can be extended to obtain a probabilistic model by associating probabilistic distribution to transitions, as shown in Section VII.

An *execution* E of an LTS is a sequence of alternating states and labels $E = s_0, \ell_0, s_1, \ell_1, \dots, s_i, \ell_i, \dots$ such that $s_0 \in I$ and $s_i \xrightarrow{\ell_i} s_{i+1}$ for all $i \geq 0$. A *computation* C of an LTS is a maximal execution, i.e., either a finite execution $s_0, \ell_0, s_1, \ell_1, \dots, s_n$ such that for all $s \in S$ and $\ell \in \Sigma$, $s_n \xrightarrow{\ell} s \notin \Delta$, or an infinite execution. A label ℓ is *enabled* (resp. *fired*) in an execution E iff there exists i s.t. ℓ is enabled in s_i (resp. $\ell_i = \ell$).

We are interested in computations of LTS which do not avoid systematically some states or transitions of the LTS, i.e., they are fair. A fairness constraint restricts the set of system computations to ones which are fair. We give below three fairness constraints used in the remainder of the paper. Let $C = s_0, \ell_0, s_1, \ell_1 \dots, s_i, \ell_i, \dots$ be a computation of an LTS.

Definition 2: C is **strong locally fair** (LF) iff for every ℓ, if ℓ is enabled in infinitely many states s_i in C, then $\ell = \ell_j$ for infinitely many j in C, i.e., $\Box\Diamond\ell$ enabled $\Rightarrow \Box\Diamond\ell$ fired.

LF constraint corresponds to the classic strong fair [14] constraint. Intuitively, it asks that if a label is enabled infinitely often in a computation, it is also executed infinitely often by the computation.

A stronger version of fairness is obtained by looking at the states in the neighborhood of a computation, i.e., states that can be reached using any transition from the states of a computation.

Definition 3: C is **strong globally fair** (GF) iff for every s and s' such that $s \rightarrow s'$, if $s = s_i$ for infinitely many i in C, then $s_j = s'$ for infinitely many j in C i.e., $s \rightarrow s' \wedge \Box\Diamond s \Rightarrow \Box\Diamond s'$.

In a GF computation C, any neighbor s' of a state s appearing infinitely often in C appears also infinitely often in C. Note that the neighbor state s' may be a direct successor of s through transitions in Δ labeled with different labels. Also, GF does not require to have s' a direct successor of s in C. The GF constraint corresponds to the fairness constraint required for the PP in [1].

Definition 4: C is **weakly fair** (WF) iff for every ℓ, if it exists j and s'_i with $s_i \xrightarrow{\ell} s'_i \in \Delta$ for any $i \geq j$, then $s_k \xrightarrow{\ell} s_{k+1}$ for infinitely many $k > j$ in C, i.e., $\Diamond\Box\ell$ is enabled $\Rightarrow$ $\Box\Diamond\ell$ is fired.

WF constraint states that if a label becomes enabled forever after some steps, then it must be fired infinitely often. The weak fairness have been well studied and the verification under weak fairness is supported by performant tools like Spin [6].

Let F be one of the fairness above ($F \in \{\text{LF, GF, WF}\}$). We use the notation $comp_F(\mathcal{T})$ for the set of F computations of the LTS $\mathcal{T}$.

From the algorithmic point of view, testing that a computation is strong fair (i.e., LF or GF) requires to compute its strongly connected components. Testing that a computation is weak fair can be done in constant time [6], i.e., very efficiently.

III. Population Protocol Model

In this section, we briefly introduce the population protocol model. More details are available in [1].

For a population of k agents, $\Pi = \{\pi_1, \dots, \pi_k\}$ denotes the set of agents. Each agent in the system is modeled as a finite state machine, representing the *program* of the agent. These programs are *uniform*: each agent executes the same finite state machine which form does not depend on the number of agents in the system. This assumption makes the model strongly *anonymous*: the agents can not store a unique identifier. When two agents *meet* each other, they may interact and perform a change in their states. The underlying network of communication is a complete graph, every pair of agents may meet. The initial state of each agent is set by the protocol. The goal of a PP is to compute by each agent the same value called the *output* of the protocol. The value to be computed is a predicate on the initial state of the protocol.

The following definition [1] indicates how to specify population protocols.

Definition 5: A **population protocol** (PP) is specified as a six-tuple $\mathcal{P} = (Q, X, Y, \iota, \omega, \delta)$ which contains a finite set Q of possible agent states, an input assignment $\iota : X \to Q$, an output assignment $\omega : Q \to Y$, and a transition relation $\delta \subseteq Q^4$.

In this definition, the initial state of an agent is fixed (using ι) by an input value in X received by the agent. In each state q, an agent outputs a value in Y given by the mapping ω. The interaction between agents is described by the relation δ: if two agents in states q_1 and q_2 *meet* each other, they change into states q_1' resp. q_2', where $(q_1, q_2, q_1', q_2') \in \delta$. We also use notation $(q_1, q_2) \to (q_1', q_2')$ for the elements of δ.

In this paper, we restrict ourselves to PP whose specifications satisfy the following conditions:

H_1: the protocol computes predicates, i.e., $Y = \{0, 1\}$;

H_2: the transition relation is non-deterministic but symmetric, i.e., $\delta \subseteq \mathbb{P}_{\leq 2}(Q) \times \mathbb{P}_{\leq 2}(Q)$ where $\mathbb{P}_{\leq 2}(Q)$ is the set of non empty subsets of Q with cardinality at most 2.

Hypotheses H_1–H_2 simplify our proofs, but do not restrict the power of the PP model. To emphasize the symmetry of transitions required by H_2, we use a special operator $\|$ instead of tuples to specify a transition in δ, i.e., $q_1 \| q_2 \to q_3 \| q_4$ (q_1 and q_2 or q_3 and q_4 may be equal). Note that a transition exists for any couple of states q_1 and q_2; when is no explicit transition with left side $q_1 \| q_2$ in δ, the default transition $q_1 \| q_2 \to q_1 \| q_2$ is applied. We denote by $U(\delta)$ the elements of δ which are not identities, i.e., $\{q_1, q_2\} \neq \{q_3, q_4\}$.

The semantics of a PP $\mathcal{P}$ is given by an LTS $\mathcal{T}_{\mathcal{P}} = \langle S, I, \Sigma, \Delta \rangle$ defined by:

- a state s (called configuration in [1]) is a mapping $\Pi \to Q$ specifying the state of each agent; then $S = 2^{\Pi \to Q}$,
- the initial set of states I, is given by the image of the input of the protocol by the function ι,
- the set of labels $\Sigma \subseteq \mathbb{P}_2(\Pi) \times \delta$, and
- the set of labeled transitions $\Delta = \{s \xrightarrow{\ell} s' \mid \exists \pi_i, \pi_j \in \Pi \text{ s.t. } t = s(\pi_i) \| s(\pi_j) \to s'(\pi_i) \| s'(\pi_j) \in \delta \text{ and } \ell = (\{i, j\}, t)\}$.

Intuitively, an interaction between two agents in the PP model is identified at the semantic level by the set of identities of the agents interacting and by the transition in δ which is applied by this interaction. The presence of an interaction in some state s of the protocol is denoted by a transition $s \to s'$ at the semantic level.

An important semantic notion for the PP is the *stable output* of a computation. To define it, we use the natural extension of ω to $Q^k \to Y^k$. A computation $C = s_0, \ell_0, s_1, \ell_1, \dots$ stably outputs some $b \in Y$ if there exists some i such that $\omega(s_j) = b^k$ for all $j \geq i$, i.e., every agent eventually outputs b and never changes its output thereafter, i.e., C satisfies $\Diamond\Box(\omega(s) = b^k)$.

One of the characteristics of the PP model is that the order in which pairs of agents interact is unpredictable. In order to model this aspect, one may imagine the presence of a scheduler which is scheduling the interactions. This scheduler may force two agents to never interact. In the presence of such scheduler, the PP has no chance to compute its goal. Thus, Angluin et al. [1] require that the scheduler allows only GF computations.

Under the GF requirement for the scheduler, a protocol of size k *stably computes* a function $f : X^k \to Y$ if, for every input $\vec{x} \in X^k$, every GF computation starting in $\iota(\vec{x})$ stably outputs $f(\vec{x})$, i.e., $\iota(\vec{x}) \wedge \text{GF} \Rightarrow \Diamond\Box(\omega(s) = f(\vec{x}))$. The important result of Angluin et al. [15] is that a predicate is stably computable by the PP model iff it can be defined as a first-order logical formula in Presburger arithmetic.

We shortly present two examples of PP that will be our running examples. For more examples, see e.g., [16].

*Example 1 (**Majority**):* Let $X = \{L, F\}$. The MAJORITY protocol stably computes the predicate $|L| > |F|$ where $|L|$ (resp. $|F|$) is the number of agents with input L (resp. F). The states of the protocol are $Q = \{L, F, 0, 1\}$, ι is the identity function, and ω maps L and 1 to 1, F and 0 to 0. The transition set δ is defined as follows:

$$\begin{array}{llcl@{\qquad}llcl} t_1: & L\|F & \to & 0\|0 & t_2: & L\|0 & \to & L\|1 \\ t_3: & F\|1 & \to & F\|0 & t_4: & 0\|1 & \to & 0\|0 \end{array}$$

[16] notices that the execution in Figure 1 is not GF because the state $(0, L, 1)$ appears infinitely often but not the state

$$\begin{tikzcd}
(L,L,F) \arrow[r, "(\{1,3\},t_1)"] & (0,L,0) \arrow[r, "(\{1,2\},t_2)"] & (1,L,0) \arrow[d, "(\{1,3\},t_4)"] \\
& (0,L,1) \arrow[u, "(\{1,3\},t_4)"] & (0,L,0) \arrow[l, "(\{2,3\},t_2)"]
\end{tikzcd}$$

Figure 1. LF execution which is not GF in the MAJORITY protocol.

$(1,L,1)$ which is its neighbor (by transition $(\{1,2\},t_2)$). However, this execution is LF because each infinitely often enabled label is fired infinitely often. From the Figure 1, it appears that this execution is also accepted by a fairness constraint which requires that any two agents meet infinitely often.

*Example 2 (**Threshold**):* Let $N \geq 2$ be an integer value and $X = \{0,1\}$. The THRESHOLD(N) protocol stably computes the function $|\iota(1)| \geq N$. The states of the protocol are $Q = \{0,\ldots,N\}$. ι maps all agents with input 1 in state 1 and all other agents in state 0. The output function ω maps a state q to the boolean value $q = N$. δ is defined as follows:

$$\begin{array}{lcccl}
t_{q,q}: & q\|q & \to & q\|q+1 & \text{if } 1 \leq q < N \\
t_{N,q}: & N\|q & \to & N\|N & \text{if } 0 \leq q < N
\end{array}$$

Notice that the size of Q (resp. δ) for this protocol is linear (resp. quadratic) on N.

IV. ABSTRACT MODEL

We introduce an abstract model for the PP that facilitates the verification task. This model has been studied in [17].

Intuitively, we abstract a state s of the $\mathcal{T}_\mathcal{P}$ by a vector of integers $\mathbf{c}$ indexed by Q such that $\mathbf{c}[q]$ is the number of agents in state q in s. This abstraction is possible due to the uniformity of the protocol: the behavior of an agent depends only on its state and not on its index.

Another intuition of this abstraction may be obtained by the translation of the PP model into a classic place/transition Petri net (PN) model. To each state in Q is associated a place in the PN, and to each transition in δ is associated a transition in the PN. Agents are represented by tokens in the PN; the firing of a transition in the PN corresponds to an interaction between agents in the PP model. Figure 2 pictures the PN obtained for the MAJORITY protocol with $k = 3$. The counting abstraction keeps track of the number of tokens in each place of the PN by associating a counter with each place (state in Q); the transitions of the PN are encoded into conditions and assignments on these counters.

Formally, the model obtained by this abstraction is a *vector addition system* model; it is equivalent (from the point of view of computation power) with the PN model, but more convenient to encode in the verification tools we consider.

Definition 6: A **vector addition aystem** (VAS) is a pair $\mathcal{A} = (\Gamma, D)$ where $\Gamma = \{c_1,\ldots,c_n\}$ is a finite set of integer

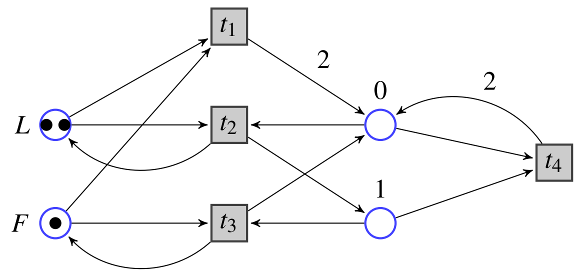

Figure 2. The Majority PP for $k = 3$ abstracted into a Petri net.

variables called *counters*, and $D \subseteq \mathbb{N}^n \times \mathbb{Z}^n$ is a finite set of *guarded translations* $\phi = (\mu, \tau)$ such that $\mu + \tau \geq \mathbf{0}$.

We represent guarded translations $(\mu,\tau) \in D$ by a constraint $\mathbf{c} \geq \mu \wedge \mathbf{c}' = \mathbf{c} + \tau$, where $\mathbf{c}$ is the vector representation of Γ, $\mathbf{c}'$ is a vector of n symbols not in Γ used to denote the new values of counters $\mathbf{c}$. The constraint $\mathbf{c} \geq \mu$ is called the *guard* (notation *guard*(ϕ)) and the constraint $\mathbf{c}' = \mathbf{c} + \tau$ is called the *translation* (notation *trans*(ϕ)). Intuitively, a guarded translation defines the condition on the values of the counters Γ in the current state, and the update of these counters to new values represented by the counters $\mathbf{c}'$.

*Example 3 (**Majority** cont.):* The VAS model representing the PN on Figure 2 is given by:

$$\begin{array}{lccc}
i & \overbrace{\qquad guard(\phi_i) \qquad} & & \overbrace{\qquad\qquad trans(\phi_i) \qquad\qquad} \\
\phi_1: & c_L \geq 1 \wedge c_F \geq 1 & \wedge & c'_L = c_L - 1 \wedge c'_F = c_F - 1 \wedge c'_0 = c_0 + 2 \\
\phi_2: & c_L \geq 1 \wedge c_0 \geq 1 & \wedge & c'_0 = c_0 - 1 \wedge c'_1 = c_1 + 1 \\
\phi_3: & c_F \geq 1 \wedge c_1 \geq 1 & \wedge & c'_1 = c_1 - 1 \wedge c'_0 = c_0 + 1 \\
\phi_4: & c_0 \geq 1 \wedge c_1 \geq 1 & \wedge & c'_1 = c_1 - 1 \wedge c'_0 = c_0 + 1
\end{array}$$

The semantics of a VAS is given by an LTS $\mathcal{T}_\mathcal{A} = \langle S, I, \Sigma, \Delta \rangle$ where:

- a state s is a valuation of counters in Γ, i.e., $s : \Gamma \to \mathbb{N}$; thus $S = 2^{\Gamma \to \mathbb{N}}$,
- the initial set of states I is a set of valuations for the counters in Γ,
- the set of labels Σ is defined by D,
- the set of transitions $\Delta = \{s \xrightarrow{\ell} s' \mid s, s' : \Gamma \to \mathbb{N}, \ell = (\mu,\tau), \text{ and } s \geq \mu \ \wedge \ s' = s + \tau\}$.

V. ABSTRACTING POPULATION PROTOCOLS

We now formalize the abstraction of the PP model into a VAS model and present the main property of this abstraction which allows us to perform the verification of the CS property on the abstract model.

A. Abstraction function

Definition 7: The **counting abstraction** α maps a PP model $\mathcal{P} = \langle Q, X, \iota, \omega, \delta \rangle$ into a VAS model $\mathcal{A} = \langle \Gamma, D \rangle$ as follows:

- $\Gamma = \{c_q \mid q \in Q\}$, and

- $D = \{(\mu_t, \tau_t) \mid \;\; \delta \ni t = q_1 \| q_2 \rightarrow q_3 \| q_4$ and
  $\mu_t = \{q \mapsto \big((q_1 \equiv q) + (q_2 \equiv q)\big)\},$
  $\tau_t = \{q \mapsto \big(\; -(q_1 \equiv q) - (q_2 \equiv q) + (q_3 \equiv q) + (q_4 \equiv q)\big)\}\}$
  where the expression $(q \equiv q')$ returns 1 if q is the same as q' and 0 otherwise.

We denote by $\alpha(t)$ the couple (μ_t, τ_t) defined above. The abstraction function α comes with a *concretization function* γ which maps back the VAS model into the PP model. Figure 3 illustrates the relation between the mapping α and γ.

From the definition of α, it results that the LTS of a VAS $\mathcal{A}$ abstracting a PP model $\mathcal{P}$ has a number of states in $O(k^{|Q|-1})$ while the LTS of $\mathcal{P}$ has $O(|Q|^k)$ states. But k varies for a fixed Q and, in general, $k \gg |Q|$. Thus, the number of states $\mathcal{T}_{\mathcal{A}}$ is smaller that the one of $\mathcal{T}_{\mathcal{P}}$ when $\mathcal{A} = \alpha(\mathcal{P})$. Also, α translates PP transitions in $U(\delta)$ into transitions of VAS with a not null translation component.

A couple of mappings (α, γ) induces a couple $(\alpha_{LTS}, \gamma_{LTS})$ defined on the LTS representing the semantics of PP and VAS models. Let $\mathcal{T}_{\mathcal{P}} = \langle S, I, \Sigma, \Delta \rangle$ be the LTS of some PP $\mathcal{P}$, and $\mathcal{T}_{\mathcal{A}} = \langle S^\#, I^\#, \Sigma^\#, \Delta^\# \rangle$ be the LTS of $\mathcal{A} = \alpha(\mathcal{P})$. The α_{LTS} mapping is defined by two mappings $\alpha_S : S \rightarrow S^\#$ and $\alpha_\Sigma : \Sigma \rightarrow \Sigma^\#$ such that:

$$\begin{array}{rcl} \alpha_S(s) & = & \{q \mapsto |\{i \mid s(i) = q\}|\} \\ \alpha_\Sigma(\{i,j\}, t) & = & \alpha(t) = (\mu_t, \tau_t) \end{array}$$

The properties of the induced mapping α_{LTS} are described by the following proposition:

Proposition 1: The components of $\mathcal{T}_{\mathcal{P}}$ and $\mathcal{T}_{\mathcal{A}}$ with $\mathcal{A} = \alpha(\mathcal{P})$ satisfy the following identities:

$$\begin{array}{ll} S^\# = \bigcup_{s \in S}\{\alpha_S(s)\}, & \Sigma^\# = \bigcup_{(\{i,j\},t) \in \Sigma}\{\alpha_\Sigma(t)\} \\ I^\# = \bigcup_{s \in I}\{\alpha_S(s)\}, & \Delta^\# = \bigcup_{s \xrightarrow{\{i,j\},t} s' \in \Delta}\{\alpha_S(s) \xrightarrow{\alpha_\Sigma(t)} \alpha_S(s')\} \end{array}$$

In the following, we remove the subscripts of the α and γ mappings if they are obvious from the context.

Because one abstract state $s^\#$ represents all the concrete states s which have the same number of agents in the same state, we have that one abstract execution $E^\#$ in $\mathcal{T}_{\mathcal{A}}$ represents several concrete executions in $E_1, E_2, \ldots$ in $\mathcal{T}_{\mathcal{P}}$. Figure 3 illustrates this fact. The concrete states situated at the same distance form the initial state are permutations of each other (relation represented by a dotted arrow in Figure 3). Also, the labels of transitions in the concrete executions abstracted by $E^\#$ correspond in their second component t.

Using the induced abstraction on LTS, we obtain that an abstract computation $C^\# = s_0^\#, \ell_0^\#, s_1^\#, \ell_1^\#, \ldots$ stably computes a predicate $f : X^k \rightarrow Y$ if there exists some i such that, for all $j \geq i$

$$\Sigma_{q, \omega(q) = f(\vec{x})} s_j^\#(c_q) \quad = \quad k \qquad (1)$$

or equivalently,

$$\Sigma_{q, \omega(q) \neq f(\vec{x})} s_j^\#(c_q) \quad = \quad 0 \qquad (2)$$

$$\begin{tikzcd} \mathcal{P} \in \mathrm{PP} \arrow[r, bend left, "\alpha"] & \mathcal{A} \in \mathrm{VAS} \arrow[l, bend left, "\gamma"] \end{tikzcd}$$

$$\begin{tikzcd} s_0 \arrow[r, "\ell_0, t_{j0}"] \arrow[dd, dotted, bend right, leftrightarrow] & s_1 \arrow[r, "\ell_1, t_{j1}"] \arrow[dd, dotted, bend right, leftrightarrow] & s_2 \arrow[r, "\ell_2, t_{j2}"] \arrow[dd, dotted, bend right, leftrightarrow] & \cdots \arrow[r] & s_i \arrow[r, "\ell_i, t_{ji}"] \arrow[dd, dotted, bend right, leftrightarrow] & \cdots \\ s_0^\# \arrow[u, "\gamma"] \arrow[d, "\gamma"] \arrow[r, "\alpha(t_{j0})"] & s_1^\# \arrow[u, "\gamma"] \arrow[d, "\gamma"] \arrow[r, "\alpha(t_{j1})"] & s_2^\# \arrow[u, "\gamma"] \arrow[d, "\gamma"] \arrow[r, "\alpha(t_{j2})"] & \cdots \arrow[r] & s_i^\# \arrow[u, "\gamma"] \arrow[d, "\gamma"] \arrow[r, "\alpha(t_{ji})"] & \cdots \\ s_0' \arrow[r, "\ell_0', t_{j0}"] & s_1' \arrow[r, "\ell_1', t_{j1}"] & s_2' \arrow[r, "\ell_2', t_{j2}"] & \cdots \arrow[r] & s_i' \arrow[r, "\ell_i', t_{ji}"] & \cdots \end{tikzcd}$$

Figure 3. Abstraction and concretization mappings.

We denote by $\alpha(f)$ the state predicate in equation 1. Then, $C^\#$ stably computes f if it satisfies $\Diamond\Box\alpha(f)$.

B. Properties of the abstraction

We are now ready to show that the verification of the correct stabilization property on the PP can be transferred into a verification of the same property in the abstract model. This result is given by the following theorem:

*Theorem 1 (**Property transfer**):* A PP $\mathcal{P}$ stably computes a function f iff its VAS abstraction $\alpha(\mathcal{P})$ stably computes $\alpha(f)$, i.e., $\mathcal{T}_{\mathcal{P}}$ satisfies $GF \Rightarrow \Diamond\Box\big(\omega(s) = f(\vec{x})\big)$ iff $\mathcal{T}_{\alpha(\mathcal{P})}$ satisfies $GF \Rightarrow \Diamond\Box\alpha(f)$

To obtain the proof of this theorem, we need the following two lemmas which relate GF computations in the LTS of the concrete and abstract models.

Proposition 2: If C is a GF computation of $\mathcal{P}$, then $\alpha(C)$ is a GF computation in $\alpha(\mathcal{P})$.
Proof: Let $C^\# = \alpha(C)$ and let $s^\#$ be a state in $C^\#$ such that $s^\#$ appears infinitely often. Recall that $s^\#$ represents a *finite* set of states in C, denoted by $\gamma(s^\#)$. Then, there is at least one state s in $\gamma(s^\#)$ which appears in C infinitely often. Let $s_1^\#$ be a state neighbor of $s^\#$, i.e., $s^\# \rightarrow s_1^\#$. Since the transitions in $\mathcal{P}$ are anonymous and symmetric, the state s has also a neighbor s_1 such that $s_1 \in \gamma(s_1^\#)$. From the hypothesis that C is a GF computation, we obtain that s_1 appears infinitely often in C, and then $s_1^\# = \alpha(s_1)$ appears infinitely often in $C^\#$.

Proposition 3: If $C^\#$ is a GF computation of $\alpha(\mathcal{P})$, then there exists $C \in \gamma(C^\#)$ s.t. C is a GF computation of $\mathcal{P}$.
Proof: Notice than not all computations in $\gamma(C^\#)$ are GF computations even if $C^\#$ is GF computation. Indeed, suppose that some state $s_i^\#$ appears infinitely often in $C^\#$, and let $z_i^\#$ be its neighbor by some transition labeled by t. If s_i is the i^{th} concrete state in some $C \in \gamma(C^\#)$, i.e., $s_i \in \gamma(s_i^\#)$, the neighbors of s_i by t are states z_i such that $z_i \in \gamma(z_i^\#)$, and $s_i \xrightarrow{\{u,v\},t} z_i$ for some couple of agents (u, v). The computation C may not execute infinitely often some of the

neighbors in $\gamma(z_i^\#)$, but still its abstraction $C^\#$ is still a GF computation.
Then, we have to prove that there exists some computation C in $\gamma(C^\#)$ that contains infinitely often any neighbor of any state s_i which appears infinitely often. To build such a sequence, we observe that the maximal number of neighbors of s_i by some transition $t \in \delta$ is a constant ξ fixed by the population size k. Thus, we build C from $C^\#$ by choosing some initial state $s_0 \in \gamma(s_0^\#)$ and by choosing agents to meet in the transitions of $C^\#$ such that all the interactions are considered by all concretizations of $s_i^\#$ which appears infinitely often in C. This choice is possible due to the symmetry property of transitions in $\mathcal{P}$.

Proof of Theorem 1: ($\Rightarrow$) Suppose that any GF computation of $\mathcal{T}_\mathcal{P}$ satisfies $\Diamond\Box\big(\omega(s) = f(\vec{x})\big)$, and consider one GF computation $C^\#$ in $\mathcal{T}_{\alpha(\mathcal{P})}$. From Proposition 2, there is some C which is a GF computation of $\mathcal{P}$ in $\gamma(C^\#)$. By hypothesis, C satisfies $\Diamond\Box\big(\omega(s) = f(\vec{x})\big)$, i.e., there exists some i such that for all $j \geq i$, s_j satisfies $\omega(s_j) = f(\vec{x})$. Since $C^\#$ has the same transitions than C, we obtain that for all $j \geq i$, $s_j^\#(c_q)$ has a strictly positive value only if $\omega(q) = f(\vec{x})$. From the definition of $\alpha(f)$ (Equation 1), it follows that $s_j^\#$ satisfies $\alpha(f)$.
($\Leftarrow$) Suppose that any GF computation of $\mathcal{T}_{\alpha(\mathcal{P})}$ satisfies $\Diamond\Box\alpha(f)$, and consider some GF computation C in $\mathcal{T}_\mathcal{P}$. From Proposition 1, $C^\# = \alpha(C)$ is a GF computation of $\mathcal{T}_{\alpha(\mathcal{P})}$. By hypothesis, $C^\#$ satisfies $\Diamond\Box\alpha(f)$, i.e., there exists some i such that for all $j \geq i$, $s_j^\#$ satisfies $\alpha(f)$. From the definition of $\alpha(f)$, it results that for any s in $\gamma(s_j^\#)$, there is no agent with state q such that $\omega(q) \neq f(\vec{x})$, which also means that $\omega(s_j) = f(\vec{x})$. Then, in computation C, all the states s_j for $j \geq i$ satisfies $\omega(s) = f(\vec{x})$.

C. Verification under weak fairness

As mentioned in Section II, the verification of a property under some strong fairness (LF or GF) constraint is more expensive than the verification under the WF constraint. Thus, we study the conditions on which the WF constraint can be used for the verification of PP. The problem of using the WF constraint is that we can obtain false negatives, since the set of WF computations includes the set of GF computations. For example, a false negative is signaled for the MAJORITY protocol because the loop shown at Example 1 is not a WF computation.

The results of the model-checking under GF and WF constraints are equal for computations which are "insensitive" to the fairness constraint. We provide two sufficient conditions on the transition system $\mathcal{T}_\mathcal{P}$ of some PP model $\mathcal{P}$ to obtain this equality. These conditions constrain the strongly connected components (SCC) of the transition system of the protocol.

Condition SCC_1**:** $\mathcal{T}_\mathcal{P}$ has only sink strongly connected components.

Condition SCC_2**:** All the SCC of $\mathcal{T}_\mathcal{P}$ have the same output value.

*Theorem 2 (**Correction of verification under WF**):* Let $\mathcal{P}$ be a PP whose $\mathcal{T}_\mathcal{P}$ satisfies SCC_1 or SCC_2. If $\mathcal{P}$ stably computes f under the WF constraint then $\mathcal{P}$ stably computes f under the GF constraint.

Proof: Recall that the correct stabilization property has the form $\Diamond\Box\phi$. Its model-checking algorithm consists in finding a point in each execution from which ϕ is satisfied in each configuration.

Condition SCC_1 means that each execution has the form of a finite sequence (without loops) followed by a SCC. For both GF and WF constraint, this kind of executions is fair. The point to be searched by the model-checking algorithm is in a state before or at the entry in the sink SCC. Thus, the result of the model-checking is given by the satisfaction of ϕ in all the states of the sink SCC.

Condition SCC_2 allows executions that loop infinitely in some SCC. The GF constraint forces the exit from any non-sink SCC, but the WF constraint does not if the SCC has at least two transitions. However, since all SCC have the same output, the result of the model-checking is the same in both cases and equal to the satisfaction of ϕ by the common output value.

Notice that the constraint SCC_1 is not satisfied by a PP model $\mathcal{P}$ which contains only identity transitions, i.e., ones not in $U(\delta)$, because identity transitions generates self loops in $\mathcal{T}_\mathcal{P}$. This check is syntactic, and can be done in constant time. For PP models where $U(\delta) = \delta$, the condition SCC_1 is not very restrictive. For example, it is satisfied by all our benchmark protocols except MAJORITY. The condition SCC_1 can be tested using classical tools on LTS [**?**], as well as the HSF-Spin [18], and extension of Spin which computes the SCC of any Spin model. Notice however that this test is as expensive as testing the strong fairness.

VI. VERIFICATION USING SPIN

The Spin model-checker [6] is one of the most efficient model-checkers for finite systems. The language for model description in Spin is called Promela. It provides intuitive constructs for modeling finite state automata extended with finite domain variables (e.g., finite counters). The general scheme of translation from our VAS model to the Promela code is given in Figure 4. The `do` loop statement contains several branches (introduced by `::`) which are chosen to be executed in a non-deterministic way; a branch is executed only if its guard (expression before $\rightarrow$) is true.

The property to be checked on the model is specified using LTL. To check an LTL formula φ on a model M, Spin executes the following two steps:

(1) *ltl2ba*: the negation of φ is translated into a Büchi automaton which thus accepts all the traces which are not satisfying φ; the size of this automaton $2^{O(|\varphi|)}$ where $|\varphi|$ is

```
do
:: guard(ϕ_1) →   trans(ϕ_1)
...    ...          ...
:: guard(ϕ_m) →   trans(ϕ_m)
od
```

Figure 4. Promela code scheme for VAS model of PP.

protocol	$\|D\|$	WF	LF
majority	4	0.057	0.021
mod3	3	0.087	0.021
broadcast	4	0.057	0.021
flock-2	3	0.043	0.021
flock-4	9	0.377	0.091
flock-8	28	—	—
threshold-2	3	0.033	0.030
threshold-4	7	—	210.515
threshold-8	14	—	—

Table I
TIME PERFORMANCES (IN SEC.) FOR THE STEP *ltl2ba*.

the number of symbols in φ, This step is called one time for each formula to be checked, so its result can be used for several models M.

(2) *check*: the product of the Büchi automaton with the input model M is done on-the-fly. ϖ is satisfied by M if there are no cycles including an an accepting state of the Büchi automaton. Otherwise, a trace of M not respecting the formula φ is provided. The time complexity of this step is then $2^{O(|\varphi|)}O(|M|)$, where $|M|$ is the sum of transitions and states in M.

The exponential size of the Büchi automaton generated by the *ltl2ba* step prevents us to encode the GF constraint in the LTL formula in order to select only GF computations. Indeed, the size of an LTL formula encoding the GF constraint is linear in the size of the LTS of M. To give an idea of the explosion of this step, we provide in Table II the execution times of the *ltl2ba* step for formulas encoding the LF and the WF constraints. These experiments have been done on an Intel i686 double core with 4GHZ and memory limited to 512MB. All experiments taking more than 30 minutes have been stopped (entries "—" in the table).

Fortunately, Spin has an option to select WF computations during the *check* step. This selection is done very efficiently by Spin. From the Theorem **??**, the result of model-checking with Spin under the WF constraint is valid for models satisfying SCC_1 or SCC_2. Thus, we have also to do the check on one of these conditions.

A. Benchmark

The PP models in our benchmark are given on Table I together with the characteristics of their VAS model: the number of transitions ($|D|$) and the number of counters ($|\Gamma|$). The *majority* and *threshold-n* protocols are our running examples. The *broadcast* protocol computes the OR predicate on the input values. The *mod3* protocol instantiate the one given in [15] to compute the predicate testing that the quotient by 3 of the number of agents with input 1 is equal to the number of agents in state 1. The *flock-n* protocols are the "flock of birds" protocol given in [13] which computes the same predicate that *threshold-n* but in a different manner.

protocol	computes	$\|D\|$	$\|\Gamma\|$
majority	$\|\iota(1)\| > \|\iota(0)\|$	4	4
mod3	$[\|\iota(1)\| / 3] = \|\omega(1)\|$	3	4
broadcast	$[\|\iota(1)\| \geq 1]$	1	2
flock-2	$\|\iota(1)\| \geq 2$	3	3
flock-4	$\|\iota(1)\| \geq 4$	9	5
flock-8	$\|\iota(1)\| \geq 8$	28	9
threshold-2	$\|\iota(1)\| \geq 2$	3	3
threshold-4	$\|\iota(1)\| \geq 4$	7	5
threshold-8	$\|\iota(1)\| \geq 8$	14	9

Table II
BENCHMARK OF PP MODELS.

protocol	SCC_1	population size k		
		11	101	1001
majority	No	0.061 (-)	0.324 (-)	0.185 (-)
mod3	Yes	0.128 (+)	0.377 (+)	28.142 (+)
broadcast	Yes	0.042 (+)	0.049 (+)	0.083 (+)
flock-2	Yes	0.055 (+)	0.250 (+)	22.748 (+)
flock-4	Yes	0.122 (+)	52.973 (+)	51.331 (+)
flock-8	Yes	0.070 (+)	59.198 (+)	57.904 (+)
threshold-2	Yes	0.045 (+)	0.199 (+)	17.554 (+)
threshold-4	Yes	0.080 (+)	40.441 (+)	40.551 (+)
threshold-8	Yes	0.881 (+)	65.458 (+)	64.450 (+)

Table III
PERFORMANCES (IN SEC.) FOR CHECKING CORRECT STABILIZATION.

B. Experimental results

Table III presents the experimental results obtained for the phase *check* on several instances of PP in our benchmark. The execution memory is limited to 512MB. The worse execution times observed for different inputs are given. The entries annotated with (+) (resp. (-)) denote a positive (resp. negative) answer of the checking phase. The column SCC_1 indicates whether the model satisfies condition SCC_1.

As expected, the MAJORITY protocol does not satisfies the condition SCC_1 and Spin returns a negative result. For $k = 3$, the execution obtained as witness for the negative result is the one shown in Figure 1. Notice that all other protocols verify condition SCC_1 and the checking times are very performant.

VII. Probabilistic verification

A. Probabilistic model

An alternative way to introduce fairness constraints in the PP model is to consider probabilistic schedulers. Angluin et al. consider this alternative in [19] and define the notion of probabilistic computation for a PP model.

*Definition 8 (**Probabilistic computation**):* A population protocol $\mathcal{P} = \langle Q, X, Y, \iota, \omega \rangle$ of population size N computes a predicate $f : X^N \longrightarrow \{0,1\}$ in w steps with error probability ε if for all $\vec{x} \in f^{-1}(\{0,1\})$, the state s reached after w steps satisfies the following properties with probability $1-\varepsilon$:

1) all agents agree on the correct output, i.e., $\forall i \in \{1,\ldots,N\}\ f(\vec{x}) = \omega(s(\pi_i))$, and
2) the previous property is also true for every state reachable from s.

*Definition 9 (**Probability of a computation**):* For a given scheduler, the probability that a PP $\mathcal{P} = \langle Q, X, Y, \iota, \omega \rangle$ computes a predicate f on input $\vec{x}$ is the probability of all computations starting in state $\iota(\vec{x})$ that stabilize with output $f(\vec{x})$.

Let us introduce now two different probabilistic schedulers. The simplest and more natural one has been considered in [1] and schedules ordered pairs of agents.

Definition 10: A **random pairing** (RP) scheduler choses, in each state of a computation, the ordered pair of agents which interact in a random independently and uniformly manner from all ordered pairs of agents.

The modeling of the RP scheduler needs the LTS of a PP because it makes its decision based on the state. It cannot be modeled on an abstract VAS model because the VAS model collapses transitions whose set of agents is the same. But working on the LTS of PP suffers from the state explosion problem when the population size increases. For example, the Majority protocol can only be verified with existing probabilistic model-checkers for population sizes less than 30 agents.

Therefore, we define a scheduler that can be used in conjunction with the counting abstraction because its decision is based on the transitions of the PP which are also transitions of the abstract VAS model.

Definition 11: The **random ruling** (RR) scheduler choses, in each state of the computation, one enabled transition $t \in \delta$ of the PP in a random uniformly and independently manner.

Note that for each state s, the set of enabled transitions is a subset of δ. Therefore, the probability of each transition to be chosen by the scheduler is greater than a strictly positive constant $(1/|\delta|)$. This fact allows us to obtain the following property as a consequence of the ergodic theorem on finite discrete Markov chains:

Proposition 4: A population protocol $\mathcal{P}$ computes a predicate f almost surely with a random pairing scheduler if and only if $\mathcal{P}$ computes f almost surely with a random ruling scheduler.

Notice that, for the non-probabilistic model, the LF constraint, which also considers the firing of transitions of $\mathcal{P}$, is not sufficient to verify the correct stabilization property for all PP, as illustrated in Example 1. In the probabilistic case, considering states (random pairing) or transitions (random ruling) is equivalent for almost sure computations of predicates.

In fact, as in the non-probabilistic case, to verify that a predicate f is computed almost surely when probabilities are strictly positive consists in searching the closed strongly connected components in the transition system. Since the LTS in the probabilistic and non-probabilistic cases are the same, we obtain the following result:

Proposition 5: A PP $\mathcal{P}$ stably computes a predicate f
$\Leftrightarrow$ $\mathcal{P}$ computes f almost surely with a RP scheduler
$\Leftrightarrow$ $\mathcal{P}$ computes f almost surely with a RR scheduler.

Hence, we can use probabilistic model-checkers to verify that $\mathcal{P}$ stably computes a predicate f even for a non randomized model of population protocols. Indeed, we have to verify that $\mathcal{P}$ computes f almost surely with some scheduler assigning strictly positive probabilities to transitions.

B. Experiments with PRISM

Prism [20], [7] is a tool for formal modeling and analysis by model-checking of systems which exhibit random or probabilistic behavior. Models are described using the PRISM language, a simple, state-based language. The property specification language incorporates the several temporal logics including LTL, as well as extensions for quantitative specifications and costs/rewards.

We encode in Prism the abstract VAS model used for Spin, without probabilities, but specified as a discrete Markov chain. Then, Prism associates to each transition enabled at some state a uniform distribution of probabilities. Based on the results of the previous section, we model-check the VAS model against the property $\Diamond\Box\alpha(f)$ with probability at least one, i.e., almost surely.

Table IV presents the experimental results obtained with Prism. The time reported at each entry is the maximal time observed on the different inputs for a given population size. Entries marked by '—' denote a timeout (fixed to 30 minutes) of the experiment. Entries marked by (*) correspond to a partial experiment where some inputs of the protocol produce a timeout.

VIII. Extensions

The previous sections demonstrate that standard verification tools may help to prove that PP stably computes some predicate. In fact, these tools may be used to verify

protocol	population size k		
	11	101	1001
majority	0.140	4.824	(*) 27.202
mod3	0.228	0.570	(*) 70.486
broadcast	0.137	0.385	14.982
flock-2	0.284	14.933	(*) 35.398
flock-4	0.661	(*) 122.240	(*)13205.884
flock-8	0.561	(*) 4.891	(*) 1030.536
threshold-2	0.167	10.242	(*) 24.455
threshold-4	0.531	(*) 34.441	—
threshold-8	0.719	—	—

Table IV
PERFORMANCES (IN SEC.) FOR MODEL-CHECKING WITH PRISM.

```
const int N; // Population size
global IN: [1..N] init 1;  // number of infected nodes
// - transition  T1: IN, NI-> IN,IN
module T1
[] (IN >= 1) & (N-IN >= 1) ->
          (2*IN*(N-IN))/(N*(N-1)):  (IN' = IN + 1)
        + 1-(2*IN*(N-IN))/(N*(N-1)): true;
endmodule
```

Figure 5. Prism code for epidemic broadcast.

other properties of population protocols. Indeed, it could be interesting to verify more general properties on states that can be expressed by LTL formulas. For example, such properties are (1) all agents are infinitely often in some state, or (2) for PP with privacy [21], to verify that the privacy of computations is ensured.

A probabilistic model-checker like Prism allows to compute the expected average time of convergence, i.e., the number of interactions that occur before convergence. We illustrate this last extension of use to compute in a random pairing scheduler model the average time of convergence for two classical problems:

- the epidemic broadcast: some information broadcasted by some agents needs to reach all agents;
- the leader election: starting from any numbers of leaders, the protocol converges to a state with at most one leader.

Notice that for an appropriate choice of probabilities for transitions, the computation of the average time of convergence can be done on the abstract VAS model. Figure 5 (resp. Figure 6) gives the Prism code of the epidemic broadcast (resp. the leader election). In both cases, we specify probabilities for transitions corresponding to the random pairing scheduler. For example, for the epidemic broadcast (Figure 5), this probability is `2*IN*(N-IN))/(N*(N-1))`, where `IN` is the number of infected agents (i.e agents already having the information) and `N` is the population size.

Angluin et al. [22] prove that, for a population size N, the

```
const int N; //Population size
const int L=N; // Number of initial leaders
global CL: [1..N] init L;
// - transition T1: L,L-> L,N
module T1
[] (CL >= 2 )  ->
           (CL*(CL-1))/(N*(N-1)): (CL' = CL -1 )
         + 1-(CL*(CL-1))/(N*(N-1)): true;
endmodule
```

Figure 6. Prism code for leader election.

expected average convergence time is:

- for the epidemic broadcast, $\Theta(N\log(N))$, and
- for the leader election, $\Theta(N^2)$.

Using Prism and its reward computation option, we obtain these results experimentally, as shown on Figures 7 and 8.

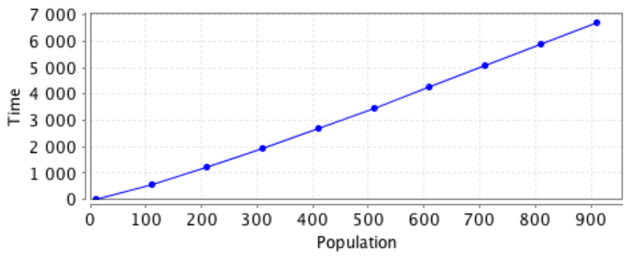

Figure 7. Expected average convergence time for epidemic broadcast.

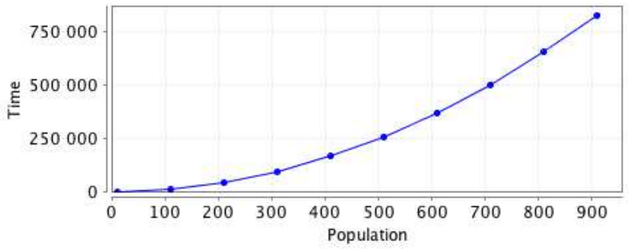

Figure 8. Expected average convergence time for leader election.

IX. CONCLUSION

We show in this paper that the counting abstraction is very useful to deal with the verification by model-checking of PP on complete graphs. Indeed, it allows to use the existing model-checking tools without facing the state explosion problem for population sizes of order of 10^3. Nevertheless, the fairness constraint of PP for stably computes property is very difficult to verify at least when using model checkers like Spin. We provide sufficient conditions to replace this fairness constraint by the weak fairness and thus to obtain scalability of verification. Moreover, since the fairness condition can be replaced by randomization, probabilistic model checkers like Prism allow to directly verify the stably computation property with good time performances.

Therefore, we recommend to use the existing model-checkers for the verification of finite instances of PP. We show that such an experience may bring to more interesting ones, e.g., the computation of the expected average time of convergence.

Finite model checkers allow only verification of PP with a fixed population size and the main extension of this work would be to deal with unbounded population protocols.

References

[1] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta, "Computation in networks of passively mobile finite-state sensors," *Distributed Computing*, pp. 235–253, Mars 2006.

[2] E. Emerson and E. Clarke, "Characterizing correctness properties of parallel programs using fixpoints," in *ICALP*, ser. LNCS, vol. 85. Springer, 1980, pp. 169–181.

[3] J.-P. Queille and J. Sifakis, "Iterative methods for the analysis of petri nets," in *Proceedings of APN*, ser. Informatik-Fachberichte, vol. 52. Springer, 1981, pp. 161–167.

[4] A. Pnueli, "The temporal logic of programs," in *FOCS*. IEEE, 1977, pp. 46–57.

[5] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 1999.

[6] G. Holzmann, *Spin model checker, the: primer and reference manual*. Addison-Wesley Professional, 2003.

[7] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with PRISM: A hybrid approach," *Journal STTT*, vol. 6, no. 2, pp. 128–142, 2004.

[8] S. German and A. Sistla, "Reasoning about systems with many processes," *J. ACM*, vol. 39, no. 3, pp. 675–735, 1992.

[9] J. Pang, Z. Luo, and Y. Deng, "On automatic verification of self-stabilizing population protocols," in *TASE*. IEEE Computer Society, 2008, pp. 185–192.

[10] R. Bakhshi, W. Fokkink, J. Pang, and J. van de Pol, "Leader election in anonymous rings: Franklin goes probabilistic," in *IFIP TCS*, ser. IFIP, vol. 273. Springer-Verlag, 2008, pp. 57–72.

[11] Y. Liu, J. Pang, J. Sun, and J. Zhao, "Verification of population ring protocols in pat," in *TASE*. IEEE Computer Society, 2009, pp. 81–89.

[12] J. Sun, Y. Liu, J. Dong, and J. Pang, "Pat: Towards flexible verification under fairness," in *CAV*, ser. Lect. Notes Comp. Sci., vol. 5643. Springer-Verlag, 2009, pp. 709–714.

[13] I. Chatzigiannakis, O. Michail, and P. Spirakis, "Algorithmic verification of population protocols," in *Proceedings of SSS*, ser. Lect. Notes Comp. Sci., vol. 6366. Springer-Verlag, 2010, pp. 221–235.

[14] N. Francez, *Fairness*. Springer, 1986.

[15] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert, "The computational power of population protocols," *Distributed Computing*, vol. 20, no. 4, pp. 279–304, 2007.

[16] J. Aspnes and E. Ruppert, "An introduction to population protocols," in *Middleware for Network Eccentric and Mobile Applications*, B. Garbinato, H. Miranda, and L. Rodrigues, Eds. Springer-Verlag, 2009, pp. 97–120.

[17] R. Karp and R. Miller, "Parallel program schemata," *Journal of Computer and System Sciences*, vol. 3, no. 2, pp. 147–195, 1969.

[18] S. Edelkamp, S. Leue, and A. Lluch-Lafuente, "Directed explicit-state model checking in the validation of communication protocols," *STTT*, vol. 5, no. 2-3, pp. 247–267, 2004.

[19] D. Angluin, J. Aspnes, and D. Eisenstat, "A simple population protocol for fast robust approximate majority," *Distributed Computing*, vol. 21, no. 2, pp. 87–102, 2008.

[20] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker, *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems,* P. Panangaden and F. van Breugel (eds.), ser. CRM Monograph Series. American Mathematical Society, 2004, vol. 23.

[21] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and E. Ruppert, "Secretive birds: Privacy in population protocols," in *OPODIS*, ser. Lecture Notes in Computer Science, E. Tovar, P. Tsigas, and H. Fouchal, Eds., vol. 4878. Springer, 2007, pp. 329–342.

[22] D. Angluin, J. Aspnes, and D. Eisenstat, "Fast computation by population protocols with a leader," *Distributed Computing*, vol. 21, no. 3, pp. 183–199, 2008.