Verification of Population Protocols

Amrita Suresh
M1 - Internship 2017-18 from February 2018 to August 2018
Supervised by Prof.Javier Esparza at the Technical University of Munich

---

General context

Population protocols are a model for distributed systems, consisting of a finite number of tiny agents. These tiny agents are indistinguishable and one can imagine them as floating around in an arbitrary fashion, with interactions occurring when two (or more) agents meet. While individual agents possess only very limited memory and computational capacity, the system as a whole is capable of performing complex distributed tasks such as leader election and majority voting. The problem of verifying the correctness of such protocols has been extensively studied, and it was found that it is at least as hard as the Petri net reachability problem.

The research problem

We address the problem of automatic verification of the correctness of such protocols. While there has been some work in the field, the question of verifying population protocols for all possible inputs is still largely unanswered. There has been some effort in using traditional model checkers to answer this question, however, it only works for protocols with fixed sizes, or a subset of all initial inputs [20, 11, 10]. Recently, it has been shown that *strongly silent protocols* can be automatically and efficiently verified with the help of constraint-solvers [9], and the logic has been implemented in the tool Peregrine [8]. However, the most succinct protocols known for some tasks are not strongly silent (we give an example in this thesis).

The purpose of my internship was to use existing ideas in the field of Petri nets in order to extend the subclass of population protocols which can be automatically verified. In doing so, it is possible to answer the verification question for a larger subclass of protocols, and verify protocols which were previously not verifiable.

Proposed contribution

Using ideas of loop acceleration, it was possible to prove the correctness of the succinct non-silent *flock-of-birds* protocol, described in [7]. Furthermore, we could automatically verify its correctness, which was previously not possible. I was also able to verify a large

number of protocol families, until a threshold, and successfully give counter-examples for the protocols that were not correct. Using ideas from the theory of well-structured transition systems, we could extend it to the verification problem of population protocols, and conjecture the correctness for a larger subclass.

Arguments supporting its validity

We were able to verify and automatically prove the correctness of a family of protocols, which was previously not automatically verified. Furthermore, we were able to extend this result to various other protocol families, and have experimental results that conjecture their correctness as well. While our solution is not as robust as the existing tool for strongly silent protocols, our method can be used for protocols outside of this subclass.

Another thing to note is that our solution uses existing theoretical results from the family of transition systems, and extends them to the verification of population protocols. Our solution makes use of the previously existing tool FASTer, described in [5], which was designed for a broader class of systems. We were able to develop a less ad-hoc notion for the automatic verification of population results using theories of flat acceleration and backwards reachability, instead of the previous notions of *layered termination* and *strong consensus* used in [9].

Summary and future work

In the course of this internship, we were able to develop a more structured notion for automatically verifying population protocols. While we may not be able to efficiently verify all protocols at this point, we can see that this characterises a large set of protocols.

The possibility of extending this result to other protocol families is an immediate follow-up. While we were able to verify the correctness of the *non-silent flock-of-birds* protocol [7], we still have not been able to characterises the set of all such protocols which can be automatically verified using this method. Furthermore, there is scope for optimising the tool in order to automatically verify protocols of larger sizes.

Acknowledgements

My stay at TUM was very fruitful largely because of the constructive and educational discussions I had with my team. I would like to thank my supervisor Prof.Esparza for his timely guidance, keen insights, and unending support. I would also like to thank my co-advisor Dr.Michael Blondin for all his helpful inputs, and his constant availability, either in person or via email. I extend my gratitude to Stefan Jaax for his prompt guidance with the code, and research, as a whole. I would also like to thank the entire chair for all the interesting and informative discussions.

I would like to thank Mrs.Claudia Link, Mrs.Imane Mimouni, and Ms.Khadidja Oudah for all the administrative support they extended. And finally, I thank Prof.Sylvain Schmitz and Prof.Philippe Schnoebelen for all the guidance before and during the internship.

1 Introduction

Population protocols are a model of distributed computation by identical, anonymous, finite-state agents of limited computational power, which work together to achieve a common task. In every step, a fixed number of agents are chosen nondeterministically to interact and update their states according to a common transition function.

Since the agents in a population protocol are anonymous, the resulting transition only depends on their current states, as there are no other identifying features. Furthermore, they are *passively mobile*, which means that the agents do not have any influence on when and with whom they interact. The model can be seen as a number of identical agents wandering around the space blindly. Structurally, population protocols can be seen as a special class of Petri nets.

Population protocols provide a simple formalism to model, e.g., networks of passively mobile sensors [3, 4], trust propagation [12], evolutionary dynamics [17], and chemical systems, under the name chemical reaction networks [19].

The population protocol model was first introduced by Angluin and others in [3]. Previously, a similar probabilistic model to study trust in a social network was introduced by Diamadi and Fischer in [12]. Since then, there has been some focus on characterising predicates computable by population protocols. In [2], it was shown that the computable predicates are exactly the semilinear predicates. These are equivalent to the predicates that can be defined in Presburger arithmetic: the first-order theory of natural numbers with addition and order.

Since the agents executing a protocol are anonymous and identical, the global state —called a *configuration* —of the protocol is completely determined by the number of agents in each state. A population protocol is said to compute a predicate on the initial state, if in all *fair* executions, all agents eventually converge to the correct value of the predicate. An execution is fair if it is finite and cannot be extended, or it is infinite and every configuration of agent states that is reachable infinitely often is reached infinitely many times along that execution.

A protocol is *well-specified* if, on every input, every fair execution eventually converges to configurations where all agents agree on a consensus value that depends only on the input. A lot of work on population protocols has focused on characterising what predicates on the input values can be computed by well-specified protocols.

A protocol is said to be correct if it behaves correctly on all the possible executions of all possible initial configurations of the protocol. Furthermore, a population protocol is parameterised, which means the initial number of agents can be unbounded even though the number of agents stay unchanged during the execution of the protocol. Hence, it is challenging to design correct and efficient protocols, and also verify that they are, indeed, correct.

In previous work, some authors have studied the automatic verification of population protocols[20, 11]. Since a protocol has a finite state space for each initial configuration, model checking algorithms can be used to verify that the protocol behaves correctly for a finite number of initial configurations. However, this technique cannot prove that the

protocol is correct for every configuration. It was shown that the problem of deciding whether a protocol computes some predicate, and the problem of deciding whether it computes a given predicate, are both decidable and at least as hard as the reachability problem for Petri nets [14]. More precisely, the problem is known to be EXPSPACE-hard, and all known algorithms for it have non-primitive recursive complexity [18].

In [8], the authors introduce PEREGRINE, the first tool for the parameterised analysis of population protocols. PEREGRINE can formally verify the correctness of protocols automatically over all of the infinitely many initial configurations using the recent approach of [9] for solving the so-called well-specification problem. However, PEREGRINE cannot verify correctness of all protocols, but only of silent protocols, a large subclass of protocols.

From results in [7], it was seen that there are efficient protocols outside of this subclass in the literature which are still yet to be verified. We look at alternative techniques to verify such protocols.

Our main result is the automatic verification of the *flock-of-birds* family of protocols, described in [9], which does not belong to the class of strongly silent protocols that can be automatically verified by PEREGRINE. The proof of correctness uses results from the theory of Petri nets. We use the theory of accelerations [16] to automatically verify the family of protocols. We rely on the semi-decision procedure described in [1] to compute the upward closure of the sets of configurations.

This report is organised as follows. Section 2 introduces some preliminaries, and the scheme of population protocols. Section 3 goes on to describe the problem we are interested in solving, clearly differentiating *silent* and *non-silent* protocols. It also gives the reader some idea about some families of protocols for which the verification of correctness can be automatically verified, and some examples where it is still unanswered. Section 4 talks about the initial approach we adopted using ideas of *upward closure* of *unstable configurations* and the incompleteness of the approach. Section 5 talks about the concept of *flatness* and *acceleration*. It formulates our automatic approach to extend the families of protocols for which we can automatically verify correctness. Finally, section 6 gives some experimental results and the relevance of our solution.

2 Preliminaries

2.1 Basic notions

Let $n \in \mathbb{N}_{>0}$. The logarithm in base b of n is denoted by $\log_b n$. When we write $\log n$, it is implicit that the base $b = 2$. We define $\text{bits}(n)$ as the set of indices of the bits occurring in the binary representation of n, e.g. $\text{bits}(9) = \{0, 3\}$ since $9 = 1001_2$. The *size* of n is the number of bits required to represent n in binary, and it is denoted by $\text{size}(n)$. We see that $|\text{bits}(n)| \leq \text{size}(n) = \lfloor \log n \rfloor + 1$.

A *multiset* over a finite set E is a mapping $M \colon E \to \mathbb{N}$. Intuitively, it is a set which allows multiple occurrences of the same element. For every element $e \in E$, $M(e)$ denotes

the number of occurrences of e in M. For example $\wr 1, 1, 4\int$ is a multiset M such that $M(1) = 2$, $M(4) = 1$ and $M(e) = 0$ for every $e \in E\backslash\{1, 4\}$. The *size* of a multiset $M \in \mathbb{N}^E$ is $|M| \stackrel{\text{def}}{=} \Sigma_{e\in E} M(e)$. The *support* of $M \in \mathbb{N}^E$ is $[\![M]\!] \stackrel{\text{def}}{=} \{e \in E : M(e) > 0\}$. Comparison is extended to multisets componentwise, i.e. $M \leq M' \stackrel{\text{def}}{\iff} M(e) \leq M'(e)$ for every $e \in E$. Addition is defined as $(M + M')(e) \stackrel{\text{def}}{=} M(e) + M'(e)$, for every $e \in E$. We define multiset difference as $(M \ominus M')(e) \stackrel{\text{def}}{=} \max(M(e) - M(e'), 0)$, for every $e \in E$.

A *finite graph* is a pair $G = (V, E)$, where V is a finite set of *nodes*, and $E \subseteq V \times V$ is a set of *edges*, such that $(u, v) \in E$ means that there is an edge from u to v. Graphs can also be directed. Therefore it may be the case that $(u, v) \in E$ but $(v, u) \notin E$. We say that there is a *path* from u to v if $v = u$ or there is a sequence of nodes $x_1, x_2, \ldots, x_n$ such that $(v, x_1) \in E, (x_1, x_2) \in E$ and so on, and $(x_n, u) \in E$.

A set $C \subseteq V$ is a *strongly connected component (SCC)* of G if C is a maximal set such that for all nodes $c, c' \in C$, there is a path from c to c' and a path from c' to c. A *bottom strongly connected component* (BSCC) of G is an SCC of G from which there are no outgoing edges. More formally, an SCC C is a BSCC if $\forall c \in C$, $\forall v \in V, (c, v) \in E \implies v \in C$. We can also lift the reachability relation to SCCs, i.e. for two SCCs C and D, we write $C \to D$ iff there exists $v \in C$ and $w \in D : (v, w) \in E$.

2.2 Protocol scheme

A *population* P over a finite set E is a multiset $P \in \mathbb{N}^E$ such that $|P| \geq 2$. The set of all populations over E is denoted by $Pop(E)$. A *k-way population protocol* is a tuple $P = (Q, T, \Sigma, I, O)$ where :

- Q is a non-empty finite set of states,
- $T \subseteq \bigcup_{2\leq i\leq k} Q^i \times Q^i$ is a set of transitions such that for every $(p, q) \in Q^2$, there exists at least a pair $(p', q') \in Q^2$ such that $((p, q), (p', q')) \in T$,
- Σ is a non-empty finite input alphabet,
- $I : \Sigma \to Q$ is the input function mapping input symbols to states,
- $O : Q \to \{0, 1\}$ is the output function mapping states to boolean values.

We call the elements of $Pop(Q)$ *configurations*. A configuration C describes a collection of identical finite-state agents with Q as the set of states, containing $C(q)$ agents in state q for every $q \in Q$, and at least two agents in total.

For a configuration, we define the output function $O\colon Pop(Q) \to \{0, 1, \bot\}$ as follows:

$$O(C) \stackrel{\text{def}}{=} \begin{cases} 1 \iff O(p) = 1 \quad \forall p \in C, \\ 0 \iff O(p) = 0 \quad \forall p \in C, \\ \bot \text{ otherwise.} \end{cases}$$

Let $t = ((p_1, p_2, \ldots, p_i), (q_1, q_2, \ldots, q_i))$ be a transition. Intuitively, t describes that i agents at states $p_1, p_2, \ldots, p_i$ may interact and move to states $q_1, q_2, \ldots, q_i$. The preset and postset of t are respectively ${}^\bullet t \stackrel{\text{def}}{=} \{p_1, p_2, \ldots, p_i\}$ and $t^\bullet \stackrel{\text{def}}{=} \{q_1, q_2, \ldots, q_i\}$. This is extended to sets of transitions as well, e.g. $T^\bullet \stackrel{\text{def}}{=} \bigcup_{t \in T} t^\bullet$. And the *pre-multiset* and *post-multiset* of t are respectively $pre(t) = \wr p_1, p_2, \ldots, p_i \int$ and $post(t) \stackrel{\text{def}}{=} \wr q_1, q_2, \ldots, q_i \int$. We say that t is *silent* if $pre(t) = post(t)$.

A transition t is *enabled* in a configuration C, if $C \geq pre(t)$. If t is executed from a configuration C, we reach the configuration $C' = (C \ominus pre(t)) + post(t)$. It is denoted by $C \xrightarrow{t} C'$. Note that, if t is silent, then $C = C'$. We write $C \xrightarrow{*} C'$ if there exists a finite number of transitions $\delta_1, ..., \delta_k$ such that $C = C_0 \xrightarrow{\delta_1} C_1 ... \xrightarrow{\delta_k} C_k = C'$. In this case, we say C' is reachable from C.

Let $\mathcal{M}$ be a set of configurations, and let t be a transition. We define:

$$pre(\mathcal{M}, t) = \{C' \colon C' \xrightarrow{*} C \text{ for some } C \in \mathcal{M}\},$$
$$pre(\mathcal{M}) = \bigcup_{t \in T} pre(\mathcal{M}, t),$$

and further,

$$pre^0(\mathcal{M}) = \mathcal{M},$$
$$pre^{i+1}(\mathcal{M}) = pre(pre^i(\mathcal{M})) \text{ for every } i \geq 0\ ,$$
$$pre^*(\mathcal{M}) = \bigcup_{i=0}^{\infty} pre^i(\mathcal{M}).$$

We can extend the same idea for the postsets, as well, with:

$$post^*(\mathcal{M}) = \bigcup_{i=0}^{\infty} post^i(\mathcal{M}).$$

2.3 Petri nets

Petri nets are similar to population protocols, but are broader and more generic models of computation.

A *Petri net* $N = (P, T, F)$ consists of a finite set P of *places*, a finite set T of *transitions*, and a flow function $F : (P \times T) \cup (T \times P) \to \mathbb{N}$. Given a transition $t \in T$, the multiset $pre(t)$ of input places of t is defined as $pre(t)(p) = F(p, t)$. Likewise, the $post(t)$ of output places of t is defined as $post(t)(p) = F(t, p)$.

A *marking* M of a net N is a multiset of places. This is equivalent to a configuration in population protocols. A transition $t \in T$ is enabled at a marking M if $pre(t) \leq M$. If it is *fired* and reaches a marking M', we denote that as $M \xrightarrow{t} M'$. Let $\sigma = t_1 \dots t_k$ be a finite sequence of transitions. We write $M \xrightarrow{\sigma} M'$ if there exist markings $M_0, \dots, M_k$ such that $M = M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots \xrightarrow{t_k} M_k = M'$. If this holds, we say that M' is *reachable* from M.

The primary difference between Petri nets and population protocols is that population protocols conserve the number of agents in an execution, whereas this is not necessarily the case for Petri nets. In fact, population protocols are a subclass of Petri nets called *conservative* Petri nets.

2.4 Stability and well-specification

An execution $C_0, C_1, \dots$ *stabilises* to b for a given $b \in \{0, 1\}$ if there exists $n \in \mathbb{N}$ such that $O(C_m) = b$ for every $m \geq n$. A configuration C is said to be *output-stable* if all agents agree on the output and continue to agree in all successor configurations, that is, $O(C) = O(C') \neq \perp$ for every configuration C' reachable from C. A configuration that is not output-stable is said to be *output-unstable*.

Since executions are infinite, instead of requiring termination, we look at convergence of the protocol. Let σ be an execution. Then by $\sigma(i) = C$, we denote that the i^{th} configuration in the execution is C .We say that an execution σ converges to an output value b if there exists $i \geq 0$ so that for all $j \geq i$, it holds that $O(\sigma(j)) = b$. We write $O(\sigma) = b$ for this. This means transitions will still occur, and the states of agents can still change, but their output values must remain the same. We also say that the computation has reached a *lasting consensus*. Convergence and stability are equivalent notions.

A protocol is *well-specified* if it computes a value for each of its infinitely many initial configurations. In other words, if every fair execution of the protocol from a given input configuration stabilises to the same value, it is said to be well-specified. More formally, a protocol is well-specified if for all initial configurations C_0, there exists $b \in O$ such that for all executions σ, if $\sigma(0) = C$, then $O(\sigma) = b$.

2.5 Upward closures

A set $U \subseteq \mathbb{N}^d$ is said to be *upward closed* if for every $u \in U, m \in \mathbb{N}^d$, we have $u \leq m \implies m \in U$. The *upward closure* of a configuration $u \in \mathbb{N}^d$ is the set $\{m \in \mathbb{N}^d | u \leq m\}$, denoted by $\uparrow u$. The upward closure of any set $M \subseteq \mathbb{N}^d$ is the set $\bigcup_{m \in M} \uparrow m$, denoted by $\uparrow M$.

Further, we have that $\uparrow M$ is the least upward closed set that contains M. Since $\mathbb{N}^d$ is well-quasi-ordered by $\leq$, for any upward closed set $U \subseteq \mathbb{N}^d$, there exists a finite set $F \subseteq U$ such that $U = \uparrow F$. This means that upward closed sets can be symbolically represented by finite sets. Such a set is called an *upward basis* (basis for short) of the upward closed set U. It is minimal for inclusion among all bases, and is called the *minimal upward basis* (minimal basis for short) of the upward closed set U.

2.6 Global fairness

It is necessary to have a formalised notion of fairness to ensure that valid computations take place in a protocol. We define *global fairness* as follows:

Definition 2.1. ***(Global fairness).*** *Let $P = (Q, T, \Sigma, I, O)$ be a population protocol. For all configurations $C, C' \in Pop(Q)$, and all executions σ, if C occurs infinitely often in σ, and $C \xrightarrow{*} C'$, then C' occurs infinitely often in σ.*

While we have the concept of global fairness, it is still important to understand the definition of a *fair execution*. We see that this is closely related to the concept of the BSCCs of the reachability graph of a configuration. An execution σ starting from configuration C is fair iff σ visits all nodes of some BSCC of C infinitely often.

To understand this idea, we first observe that σ cannot visit configurations in two different BSCCs infinitely often, since by definition, there are no outgoing edges from a BSCC. Similarly, σ cannot visit configurations from multiple SCCs infinitely often, since there are no cycles between SCCs. Thus, σ must be a proper subset of a BSCC, or a subset of a SCC.

Let us assume that σ is a proper subset of a SCC. Then, there exists a configuration v in the SCC such that $v \notin \sigma$. On the other hand, if we assume that σ is a SCC, by definition, there exists some node v outside the SCC that is reachable from the SCC. Hence, in both cases, there exists a configuration v that is reachable infinitely often, but is never reached. This is a contradiction to the fairness condition. Thus, σ must be a proper subset of a BSCC. And we know that if σ is a proper subset of a BSCC, it can only visit the nodes inside the BSCC, and it does visit them all infinitely often. Thus, the fairness condition is preserved.

3 Silent and non-silent protocols

Protocols in literature have been classified into two types, *silent* and *non-silent*. Silent, or *terminating* protocols, are protocols where the protocol stabilises to a single configuration after a finite period of time, in all fair executions. Non-silent protocols are protocols where the protocol need not stabilise to a single configuration, but reaches a *strongly connected component* after finite amount of time.

Silent protocols were introduced in [13]. A protocol is silent if communication between agents eventually stops, i.e, every fair execution eventually stays in the same configuration forever. It is important to note that there can be well-specified protocols which are not silent, as long as the configurations that the executions keep alternating between are in a consensus, with the same output.

Formally, we say that an execution $C_0C_1\ldots$ is *silent* if there exists a configuration C and an $n \in \mathbb{N}$, such that $C_i = C$ for all $i \geq n$. We say that a configuration C is *terminal* if $C \xrightarrow{*} C'$ implies $C = C'$. In other words, if every transition enabled at C is silent, then C is terminal. A population protocol P is silent if every fair execution of P is *silent*, regardless of the initial configuration. A protocol that is well-specified and silent is a

WS^2*protocol*, and all these protocols are represented by the set WS^2, described first in [9].

The protocols in the WS^2 class are exactly the protocols satisfying the following two properties:

- TERMINATION : for every configuration C, there exists a terminal configuration C' such that $C \xrightarrow{*} C'$.
- CONSENSUS: for every initial configuration C_0, there exists $b \in \{0, 1\}$ such that every terminal configuration reachable from C_0 has an output b.

In [9], the authors went on to further refine the class of silent well-specified protocols to describe a class defined as WS^3. The tool developed in [8], PEREGRINE, can successfully verify silent protocols, which belong to the WS^3 class, but cannot verify any protocol that is non-silent.

We now look at an example of a population protocol. The *threshold predicates* are of the form $a_1x_1 + a_2x_2 + \cdots + a_dx_d < c$, with parameters $a_1, \ldots, a_d, c \in \mathbb{Z}$, and $x_1, \ldots, x_d \in \mathbb{N}$ being the input variables. We look at a special case of threshold predicates is known as the *flock-of-birds* predicate, described originally as follows. Consider a flock of birds, where each bird either has normal or elevated temperature, and fix some natural number n. The question is: are there more than n birds with elevated temperature?

We look at two different population protocols that can compute this predicate. The first one was defined as *Threshold* in [11]. It is intuitively described as follows. When two agents in the same state k meet, one of them assumes the value of the immediate successor $k + 1$, while the other is left unchanged k. However, if the value n is reached, the agents with the value n will try to form a consensus by interacting with every agent and converting its state to n.

Example 3.1. ***(Silent protocol to compute*** $x \geq n$***).***

The states are $Q_n \stackrel{def}{=} \{0, 1, \ldots, n\}$*, and the initial state is* $I_n \stackrel{def}{=} \{1\}$*.*
The output mapping is defined as $O_n(n) \stackrel{def}{=} 1$*, and* $O_n(q) \stackrel{def}{=} 0$*,* $\forall q \neq n$*.*

For every state $q \in Q_n$*, let* $val(q)$ *denote the number* q *stands for, i.e.* $val(\mathbf{0}) = 0, val(\mathbf{n}) = n$ *and* $val(\mathbf{i}) = i$*, for every* $0 \leq i \leq n$*. For any configuration* C*,* $val(C) = \Sigma_{q \in Q}\ val(q).C(q)$*.*
The set of transitions T_n *is the union of two sets* T_1 *and* T_2*, and are as follows:*

$$T_1 : \mathbf{i}, \mathbf{i} \mapsto \mathbf{i} + \mathbf{1}, \mathbf{i} \qquad \text{for every } 0 < i < n.$$
$$T_2 : \mathbf{n}, q \mapsto \mathbf{n}, \mathbf{n} \qquad \text{for every } q \in Q_n.$$

As we can see, T_1 is the set of all transitions that *add up* to reach n, if possible. And once n is reached, T_2 *attracts* all the other agents to n to form a consensus. We can see that eventually, all the agents will reach n if there are at least n agents to begin with, otherwise all of them will assume one value less than or equal to the sum of the initial agents. It is, therefore, a silent protocol.

The representation of the protocol is seen in *Figure 1*, for the predicate $x \geq 3$. The transitions from the set T_1 are labelled as T_1^1 and T_1^2. The transitions from the set T_2 are labelled as T_2^0, T_2^1 and T_2^2. The states representing $0, 1, 2, 3$ are S_0, S_1, S_2, S_3 respectively, and we represent the fact that S_3 has a different output than the other states by giving the state a different colour.

An example of an execution with the initial configuration being $C_0 = (0, 3, 0, 0)$ in this protocol is as follows.
$(0,3,0,0) \xrightarrow{T_1^1} (0,2,1,0) \xrightarrow{T_1^1} (0,1,2,0) \xrightarrow{T_1^2} (0,1,1,1) \xrightarrow{T_2^1} (0,0,1,2) \xrightarrow{T_2^2} (0,0,0,3)$, which is a terminal configuration.

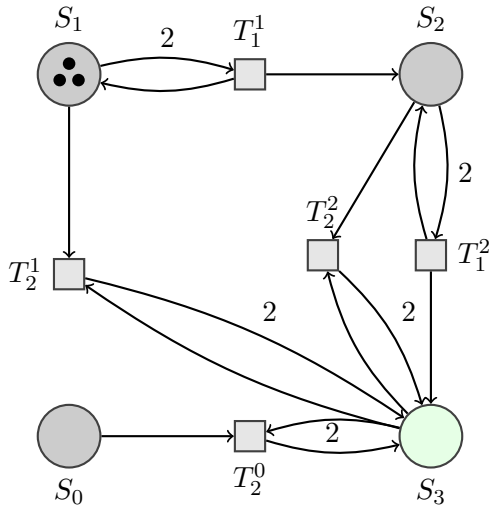

Figure 1: Flock of birds protocol to compute $x \geq 3$

However, not all protocols which are of small sizes are silent. In [7], it was found that there exists a succinct protocol to compute the predicate $x \geq n$ with $O(\log n)$ states which is not silent.

Intuitively, this protocol tries to reach a consensus by trying to reach the exact binary representation of the threshold value. Since the transitions in T_1 and T_2 are reversible, the protocol can reach all possible binary representations, and hence, it will eventually reach the representation that will enable T_3. However, since the transitions in T_1 and T_2 are reversible, if T_3 is never enabled, the protocol can alternate between various representations. Hence, it is non-silent if the initial value is less than the threshold. It is described formally in the example below.

Example 3.2. ***(Non-silent protocol to compute*** $x \geq n$***).***
The states are $Q_n \stackrel{def}{=} \{0, 2^0, \ldots, 2^{size(n)}, n\}$*, and the initial state is* $I_n \stackrel{def}{=} \{2^0\}$*.*
The output mapping is defined as $O_n(n) \stackrel{def}{=} 1$*, and* $O_n(q) \stackrel{def}{=} 0$*,* $\forall q \neq n$*.*

For every state $q \in Q_n$*, let* $val(q)$ *denote the number* q *stands for, i.e.* $val(\mathbf{0}) = 0, val(\mathbf{n}) = n$ *and* $val(\mathbf{2^i}) = 2^i$*, for every* $0 \leq i \leq size(n)$*. For any configuration* C*,*

$val(C) = \Sigma_{q \in Q}\ val(q).C(q)$.
T_n is the union of two sets T_n^1 and T_n^2. Intuitively, T_n^1 includes all the transitions that change the representation of a number m to other representations of the number m. The transitions in T_n^1 are:

$$\begin{aligned} &T_1 : \mathbf{2^i}, \mathbf{2^i} \mapsto \mathbf{2^{i+1}}, \mathbf{0} && \text{for every } 0 \leq i < size(n). \\ &T_2 : \mathbf{2^{i+1}}, \mathbf{0} \mapsto \mathbf{2^i}, \mathbf{2^i} && \text{for every } 0 \leq i < size(n). \\ &T_3 : \wr \mathbf{2^i} : i \in bits(n) \int \mapsto \underbrace{\mathbf{n}, \ldots, \mathbf{n}.}_{|bits(n)| copies} \end{aligned}$$

T_n^2 has transitions in which agents in the state $\mathbf{n}$ interact with other agents to reach a consensus. The transitions in T_n^2 are:

$$T_4 : \mathbf{n}, q \mapsto \mathbf{n}, \mathbf{n} \qquad \text{for every } q \in Q_n.$$

This protocol is not silent if $m < n$, since agents can interact and go from one representation to other representations of m. Hence, we cannot use PEREGRINE to verify this family of protocols.

An example of a fair execution with the initial configuration being $C_0 = (0, 2, 0, 0)$ in the protocol to check the predicate $x \geq 3$ is as follows.
$(0, 2, 0, 0) \xrightarrow{T_1^1} (0, 0, 1, 0) \xrightarrow{T_2^1} (0, 2, 0, 0) \xrightarrow{T_1^1} (0, 0, 1, 0) \xrightarrow{T_2^1} \ldots$ Hence, it is non-silent.

4 Unstable population protocols

The first approach used to address the problem of the verification of non-silent protocols uses the backwards reachability algorithm in order to ensure that every initial configuration would eventually reach stable configurations of the same opinion in all possible executions. Observe that stability is *downward closed*. In other words, every reachable configuration from a stable configuration C is stable, and of the same opinion as C. Conversely, the set of unstable configurations is *upward closed*. We use this property to find the basis for the set of all unstable configurations.

In order to find the finite basis for the set of all output-unstable configurations U, we define two sets of configurations $\mathcal{M}_1$ and $\mathcal{M}_2$ as follows:
$\mathcal{M}_1$ is the set of all configurations such that exactly two agents are present, and both of them have conflicting outputs:

$$\mathcal{M}_1 = \{C \colon |C| = 2 \wedge O(C) = \perp\}.$$

$\mathcal{M}_2$ is the set of all configurations C such that there exists a transition t and $C \xrightarrow{t} C'$ where C and C' both form a consensus, but of different outputs:

$$\mathcal{M}_2 = \{C : C \to C' \wedge O(C) \neq \perp \wedge O(C') \neq \perp \wedge O(C) \neq O(C')\}.$$

Let $\mathcal{M} = \mathcal{M}_1 \cup \mathcal{M}_2$. We now have the following lemma.

Lemma 4.1. *$pre^*(\uparrow \mathcal{M})$ is the set of all output-unstable configurations.*

Proof. Let us assume to the contrary that there exists a configuration C, which is output-unstable, but not present in $pre^*(\uparrow \mathcal{M})$. Any configuration that is output-unstable falls into one of the following two categories:

a) The configuration has two states with conflicting outputs, $O(C) = \perp$, or the configuration is in a consensus but can lead to a different configuration that is in a conflict, i.e. $O(C) \neq \perp$ but there exists C' such that $C \xrightarrow{*} C'$ and $O(C') = \perp$.

b) The configuration is in a consensus but eventually goes to a configuration that is in a consensus of the opposite output, i.e. $O(C) \neq \perp$ but there exists C' such that $C \xrightarrow{*} C'$ and $O(C') \neq \perp$ and $O(C) \neq O(C')$.

For case (a), any configuration C such that $O(C) = \perp$ is in $\uparrow \mathcal{M}$, because $C \in \uparrow \mathcal{M}_1 \subseteq \uparrow \mathcal{M} \subseteq pre^*(\uparrow \mathcal{M})$. So, if $O(C) = \perp$, then $C \in pre^*(\uparrow \mathcal{M})$. If $O(C) \neq \perp$, but there exists C' such that $O(C') = \perp$ and $C \xrightarrow{*} C'$, we have $C' \in pre^*(\uparrow \mathcal{M})$, and $C \in pre^*(C')$, hence, $C \in pre^*(\uparrow \mathcal{M})$. Thus, we have a contradiction.

For case (b), we have $C \to C_1 \to ... \to C'$, and $O(C) \neq O(C') \neq \perp$. Let i be the first index such that $O(C_i) \neq O(C)$. If $O(C_i) = \perp$, then $C_i \in \uparrow \mathcal{M}$ and hence, $C \in pre^*(\uparrow \mathcal{M})$. Otherwise, from the definition of $\mathcal{M}_2$, $C_{i-1} \in \mathcal{M}_2$. Thus, $C \in pre^*(\uparrow \mathcal{M})$.

Using the result above, we can see that we can compute the basis of the output-unstable elements, using backwards reachability. The general idea of the backwards reachability algorithm, introduced in [1], is to start at the target configurations to be covered (in this case, the set $\mathcal{M}$), and repeatedly compute the (minimal) predecessor configurations that reach this set of target configurations by application of one of the transitions. All such minimal configurations are added to the set of target configurations, and the process is repeated until a fixed point is reached.

Algorithm 1 Backwards reachability algorithm

Input:

Population protocol $P = (Q, T, \Sigma, I, O)$ and a set of target markings $\mathcal{M}$.

1: $M := \mathcal{M}$;
2: **while** $(B := preUpward(M) \setminus \uparrow M) \neq \emptyset$ **do**: ▷ $preUpward$ computes the minimal predecessor configurations for each element in M
3: $mergeUpward(B, M)$; ▷ $mergeUpward$ merges the sets B and M and retains the minimal elements in M
4: **end while**
5: **return** M;

Now that we have the basis for the unstable elements, we can ensure that no initial configuration will reach two different stable configurations of opposing outputs in two separate executions. However, we see that this is not sufficient to prove well-specification, since it is possible that the protocol ends up in a bottom component where there are

configurations which are not in a consensus. Our algorithm does not check for this possibility. Hence, we see that the notion of backwards reachability to get the basis for the output-unstable elements does not solve the entire problem of well-specification.

5 Automatic Verification using Acceleration

We now look at the concept of loop acceleration, introduced in [6], to try and prove the correctness problem for a subclass of population protocols.

Definition 5.1. ***(Flat protocol).*** *A population protocol P is* flat *if $\exists w_1^\star w_2^\star \dots w_n^\star = \pi$, such that $w_i \in T^*$. Then for all configurations C reachable from any initial configuration C_0, $C_0 \xrightarrow{\pi} C$.*

It was proved in [6] that every Petri net whose reachability set is semilinear is flat. Using the tool FASTer developed in [5] by Leroux and others, we were able to verify correctness for a non-silent protocol that was previously not known to be verified.

In [7], the authors showed that any predicate of the kind $x \geq n$ has an equivalent protocol of size $O(\log n)$. This protocol is described in *Example* 3.2. However, this protocol is non-silent, and hence, can not be automatically verified by PEREGRINE. We can use acceleration techniques introduced previously and prove that this protocol is correct.

Before we prove correctness of the protocol, we first define the relation between two transitions. We say a transition t_i is *independent* of a transition t_j if no agent in the pre-set of t_i is in the same state as any agent in the post-set of t_j. In other words, t_i is independent of t_j if t_i can be fired before t_j, or ${}^\bullet t_i \cap t_j^\bullet = \emptyset$. We abuse the notation slightly below, and by T_1^*, we mean $t_1^* t_2^* \dots t_n^*$, where $t_i \in T_1$ and $t_i \colon 2^i, 2^i \mapsto 2^{i+1}, 0$. We extend this notation to T_2^* and T_4^*.

Lemma 5.1. *The protocol given in Example 3.2 is flat, and if $\exists C, C'$ such that $C \xrightarrow{*} C'$, then $C \xrightarrow{\pi} C'$, where $\pi = T_1^* T_2^* T_3 T_4^*$.*

Proof. We make the following observations.

Firstly, we see that within the set T_1, all the transitions of the form $2^i, 2^i \mapsto 2^{i+1}, 0$ are independent of all the transitions $2^j, 2^j \mapsto 2^{j+1}, 0$, if $i < j$. In other words, transitions involving states of lower indices can be fired before the transitions of higher indices. This can be observed in the converse fashion in case of transitions in T_2. And we see, in case of T_4, that all transitions can be fired independently of one another.

Next, we prove that transitions in the set T_1 can be fired in the beginning, or can be omitted from the sequence altogether, using induction. Let us assume that we can reach a configuration C' from C by an arbitrary sequence of transitions. Let us assume that transitions from the set $T \backslash T_1$ have been fired, followed by a transition t_i, such that $t_i \in T_1$. Let t_i be $2^i, 2^i \mapsto 2^{i+1}, 0$.

Now we see that there are two possibilities. Either there were two agents in state 2^i initially, or they were the result of some transitions in the course of the execution.

In the first case, we can just execute this transition before the transitions preceding it, and we would still arrive at the same configuration. In the second case, we observe that the only transition which could generate agents in state 2^i is the transition t'_i from T_2 which is $2^{i+1}, 0 \mapsto 2^i, 2^i$. However, in this case, we can omit both t_i and t'_i from the sequence and still arrive at the same configuration. We can inductively go through the transition sequence in this fashion and eliminate or shift the transitions in T_1 to the beginning. Hence, we see that all the transitions in T_1 can be fired initially or completely omitted.

Now, we observe that T_2 is independent of T_3 and T_4, and T_3 is independent of T_4. Hence, we can fire execute the transitions in T_2, followed by the transition in T_3 and finally execute all the transitions in T_4.

Thus, we can reach any configuration C' from C through π, if we can reach it otherwise. Hence, the protocol supports flat acceleration. □

Now that we see that the protocol supports flat acceleration, we run the algorithm below to check its correctness. The pseudocode for the algorithm is given in *Algorithm* 2. We define $init_0$ to be the set of all initial configurations for which the predicate $\mathbb{P}$ evaluates to 0. We define *basicUnstable* as the set of all configurations such that there is at least one agent with value 0 and one agent with value 1. $consensus_0$ is the set of all configurations where there is at least one agent with output 0, and no agent with output 1. We can extend the definitions for the other terms in the algorithm by swapping 0 with 1 and vice-versa. $\mathbb{P}(\mathbb{I})$ is the predicate for the initial configurations in the protocol P.

Algorithm 2 Pseudocode for checking correctness of protocols.

Input:

Population protocol $P = (Q, T, \Sigma, I, O)$ and a predicate $\mathbb{P}(\mathbb{I})$.

1: $init_0 := \{P.initialStates >= 2 \;\&\&\; P.nonInitialStates = 0 \;\&\&\; \mathbb{P}(\mathbb{I}) = False\}$;
2: $init_1 := \{P.initialStates >= 2 \;\&\&\; P.nonInitialStates = 0 \;\&\&\; \mathbb{P}(\mathbb{I}) = True\}$;
3: $basicUnstable := \{P.trueStates >= 1 \;\&\&\; P.falseStates >= 1\}$;
4: $consensus_0 := \{P.trueStates = 0 \;\&\&\; P.falseStates >= 1\}$;
5: $consensus_1 := \{P.trueStates >= 1 \;\&\&\; P.falseStates = 0\}$;
6: $stable_0 := (!pre^*(basicUnstable))\&\&consensus_0$;
7: $stable_1 := (!pre^*(basicUnstable))\&\&consensus_1$;
8: $postSpec_0 := post^*(init_0)$;
9: $postSpec_1 := post^*(init_1)$;
10: $preStable_0 := pre^*(stable_0)$;
11: $preStable_1 := pre^*(stable_1)$;
12: **if** $\{!isSubset(postSpec_0, preStable_0)||!isSubset(postSpec_1, preStable_1)\}$ **then**
13: **return** false;
14: **else**
15: **return** true;

We see that the above protocol can be verified correct by FASTer, since the pre^* and $post^*$ both support flat acceleration. We now prove that it is sufficient to check if $post^*(init_0) \subseteq pre^*(stable_0)$ and $post^*(init_1) \subseteq pre^*(stable_1)$ to verify correctness of the protocol.

Lemma 5.2. *If we know that* $post^*(init_0) \subseteq pre^*(stable_0)$ *and* $post^*(init_1) \subseteq pre^*(stable_1)$*, we can say the protocol is correct.*

Proof. We define a protocol to be correct when it behaves correctly on all executions of all possible configurations. We first prove well-specification, and then extend the result to correctness.
Let us assume to the contrary, that the protocol is ill-specified. There can be two cases for that scenario. The first possibility is that there is some bottom strongly connected component in the protocol that has no opinion. Let us assume we can reach a configuration C in such a bottom component from an initial configuration. Without loss of generality, we assume this configuration is in $init_1$. By our result, $post*(init_1) \subseteq pre^*(stable_1)$, hence, $C \in pre^*(stable_1)$. In other words, there is an execution in C that leads to a stable configuration. This is a contradiction because we assumed C to be in a bottom configuration of no consensus. Hence, this cannot occur.
The second possibility is that there exists a configuration C' in the protocol that can reach a bottom component of two different opinions in two different executions. In other words, $post^*(C') \subseteq pre^*(stable_1)$ and $post^*(C') \subseteq pre^*(stable_1)$. But then we have a contradiction because $\exists C_0 \in stable_0 : C_0 \in post^*(C')$ and then we have, $C_0 \in pre^*(stable_1)$ which is a contradiction to the definition of a stable configuration.
Thus, it is sufficient to prove $post^*(init_0) \subseteq pre^*(stable_0)$ and $post^*(init_1) \subseteq pre^*(stable_1)$ to prove well-specification. And it follows from our definition of initial sets $init_0$ and $init_1$, that it reaches the correct consensus when it is well-specified. Hence, the protocol is correct. □

6 Experimental results

Apart from the non-silent protocol described in section 3, we evaluated the algorithm on a set of benchmarks: the remainder protocols of [4], the majority protocol of [2], the broadcast protocol of [11] and three versions of the flock of birds protocol from [10, 11, 7]. We checked the parametric protocols for increasing values of their primary parameter until we reached a timeout.

All experiments were performed on the same machine equipped with an 1,8 GHz Intel Core i5 and 8 GB of RAM. The time limit was set to 30 minutes. The results are shown in the table below. In all examples, we were able to terminate successfully for the smaller protocols in the family.

Below is the table for the verification of the protocol families mentioned above, and the performance of FASTer for the smaller families .

Protocol	v_{max}	$\|Q\|$	$\|T\|$	Time
Majority		4	4	1
Broadcast		2	1	0.8
Remainder	2	4	5	4
(modulo 1)	3	5	4	78
Flock of birds [10]	3	4	6	8
	4	4	5	1
	8	5	7	16
Flock of birds [11]	3	4	10	1
	4	5	17	0.9
	5	6	26	2
Flock of birds [7]	3	4	3	2
	4	4	4	1

Table 1: Results of the experimental evaluation of FASTer [5] on various protocols, where v_{max} denotes the parameter if any, $|Q|$ and $|T|$ denote the number of states and the number of transitions in the population protocol, and the last column is the amount of time required to verify the correctness of the protocol in seconds.

However, since the tool seems to be successful in verifying correctness for small numbers, it may be possible that all these protocols are, indeed, flat. This has yet to be formally verified, but the results look assuring. Hence, we conjecture that every Presburger definable predicate must have an equivalent population protocol which is flat.

7 Conclusion

We initiated the study of automatic verification of correctness of non-silent protocols. Previous methods were only known for a subclass of protocols known as *strongly silent protocols*. Using ideas of acceleration and flatness, we have been able to verify the correctness of the *flock-of-birds* protocol as described in [7].

There are many open questions. We conjecture that all strongly silent protocols have an equivalent protocol that is flat. Another intriguing question is whether all such protocols are succinct. In other words, we are still yet to explore whether these flat protocols would be comparable in size to the protocols in the WS^3 class [9]. The possibility to enhance the flat protocol model to experimentally verify a larger threshold is also another avenue that can be explored. Finally, it will be interesting to see if there are any protocols which cannot be succinctly characterised by an equivalent flat protocol.

References

[1] Parosh Aziz Abdulla et al. "Algorithmic Analysis of Programs with Well Quasi-ordered Domains". In: *Information and Computation* 160.1 (2000), pp. 109–127. ISSN: 0890-5401. DOI: https://doi.org/10.1006/inco.1999.2843. URL: http://www.sciencedirect.com/science/article/pii/S0890540199928432.

[2] Dana Angluin, James Aspnes, and David Eisenstat. "Stably Computable Predicates Are Semilinear". In: *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*. PODC '06. Denver, Colorado, USA: ACM, 2006, pp. 292–299. ISBN: 1-59593-384-0. DOI: 10.1145/1146381.1146425. URL: http://doi.acm.org/10.1145/1146381.1146425.

[3] Dana Angluin et al. "Computation in Networks of Passively Mobile Finite-state Sensors". In: *Proceedings of the Twenty-third Annual ACM Symposium on Principles of Distributed Computing*. PODC '04. St. John's, Newfoundland, Canada: ACM, 2004, pp. 290–299. ISBN: 1-58113-802-4. DOI: 10.1145/1011767.1011810. URL: http://doi.acm.org/10.1145/1011767.1011810.

[4] Dana Angluin et al. "Computation in networks of passively mobile finite-state sensors". In: *Distributed Computing* (Mar. 2006), pp. 235–253.

[5] Sébastien Bardin et al. "FAST: acceleration from theory to practice". In: *International Journal on Software Tools for Technology Transfer* 10.5 (Oct. 2008), pp. 401–424. ISSN: 1433-2787. DOI: 10.1007/s10009-008-0064-3. URL: https://doi.org/10.1007/s10009-008-0064-3.

[6] Sébastien Bardin et al. "Flat Acceleration in Symbolic Model Checking". In: *Automated Technology for Verification and Analysis*. Ed. by Doron A. Peled and Yih-Kuen Tsay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 474–488. ISBN: 978-3-540-31969-6.

[7] Michael Blondin, Javier Esparza, and Stefan Jaax. "Large Flocks of Small Birds: on the Minimal Size of Population Protocols". In: *35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*. Ed. by Rolf Niedermeier and Brigitte Vallée. Vol. 96. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 16:1–16:14. ISBN: 978-3-95977-062-0. DOI: 10.4230/LIPIcs.STACS.2018.16. URL: http://drops.dagstuhl.de/opus/volltexte/2018/8511.

[8] Michael Blondin, Javier Esparza, and Stefan Jaax. "Peregrine: A Tool for the Analysis of Population Protocols". In: *Computer Aided Verification*. Ed. by Hana Chockler and Georg Weissenbacher. Cham: Springer International Publishing, 2018, pp. 604–611. ISBN: 978-3-319-96145-3.

[9] Michael Blondin et al. "Towards Efficient Verification of Population Protocols". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing*. PODC '17. Washington, DC, USA: ACM, 2017, pp. 423–430. ISBN: 978-1-4503-

4992-5. DOI: 10.1145/3087801.3087816. URL: http://doi.acm.org/10.1145/3087801.3087816.

[10] Ioannis Chatzigiannakis, Othon Michail, and Paul G. Spirakis. "Algorithmic Verification of Population Protocols". In: *Stabilization, Safety, and Security of Distributed Systems*. Ed. by Shlomi Dolev et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 221–235. ISBN: 978-3-642-16023-3.

[11] J. Clement et al. "Guidelines for the Verification of Population Protocols". In: *2011 31st International Conference on Distributed Computing Systems*. June 2011, pp. 215–224. DOI: 10.1109/ICDCS.2011.36.

[12] Zoë Diamadi and Michael J. Fischer. "A simple game for the study of trust in distributed systems". In: *Wuhan University Journal of Natural Sciences* 6.1 (Mar. 2001), pp. 72–82. ISSN: 1993-4998. DOI: 10.1007/BF03160228. URL: https://doi.org/10.1007/BF03160228.

[13] Shlomi Dolev, Mohamed G. Gouda, and Marco Schneider. "Memory requirements for silent stabilization". In: *Acta Informatica* 36.6 (Oct. 1999), pp. 447–462. ISSN: 1432-0525. DOI: 10.1007/s002360050180. URL: https://doi.org/10.1007/s002360050180.

[14] Javier Esparza et al. "Verification of population protocols". In: *Acta Informatica* 54.2 (Mar. 2017), pp. 191–215. ISSN: 1432-0525. DOI: 10.1007/s00236-016-0272-3. URL: https://doi.org/10.1007/s00236-016-0272-3.

[15] John Hopcroft and Jean-Jacques Pansiot. "On the reachability problem for 5-dimensional vector addition systems". In: *Theoretical Computer Science* 8.2 (1979), pp. 135–159. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(79)90041-0. URL: http://www.sciencedirect.com/science/article/pii/0304397579900410.

[16] Jerome Leroux. "Presburger Vector Addition Systems". In: *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 23–32. ISBN: 978-0-7695-5020-6. DOI: 10.1109/LICS.2013.7. URL: http://dx.doi.org/10.1109/LICS.2013.7.

[17] P. A. P. Moran. "Random processes in genetics". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 54.1 (1958), 60fffdfffdfffd71. DOI: 10.1017/S0305004100033193.

[18] Sylvain Schmitz. "The Complexity of Reachability in Vector Addition Systems". In: *ACM SIGLOG News* 3.1 (Feb. 2016), pp. 4–21. ISSN: 2372-3491. DOI: 10.1145/2893582.2893585. URL: http://doi.acm.org/10.1145/2893582.2893585.

[19] David Soloveichik et al. "Computation with finite stochastic chemical reaction networks". In: *Natural Computing* 7.4 (Dec. 2008), pp. 615–633. ISSN: 1572-9796. DOI: 10.1007/s11047-008-9067-y. URL: https://doi.org/10.1007/s11047-008-9067-y.

[20] J. Sun et al. "Integrating Specification and Programs for System Modeling and Verification". In: *2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering*. July 2009, pp. 127–135. DOI: 10.1109/TASE.2009.32.