Characterizing Consensus in the Heard-Of Model

A. R. Balasubramanian
Technical University of Munich, Germany

Igor Walukiewicz
CNRS, LaBRI, University of Bordeaux, France

Abstract

The Heard-Of model is a simple and relatively expressive model of distributed computation. Because of this, it has gained a considerable attention of the verification community. We give a characterization of all algorithms solving consensus in a fragment of this model. The fragment is big enough to cover many prominent consensus algorithms. The characterization is purely syntactic: it is expressed in terms of some conditions on the text of the algorithm.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Software and its engineering → Software verification; Theory of computation → Logic and verification

Keywords and phrases consensus problem, Heard-Of model, verification

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.9

Related Version A full version of this paper is available at `https://arxiv.org/abs/2004.09621`.

Funding *A. R. Balasubramanian*: UMI ReLaX, ERC Advanced Grant 787367 (PaVeS).
Igor Walukiewicz: ANR FREDDA ANR-17-CE40-0013.

1 Introduction

Most distributed algorithms solving problems like consensus, leader election, set agreement, or renaming are essentially one iterated loop. Yet, their behavior is difficult to understand due to unbounded number of processes, asynchrony, failures, and other aspects of the execution model. The general context of this work is to be able to say what happens when we change some of the parameters: modify an algorithm or the execution model. Ideally we would like to characterize the space of all algorithms solving a particular problem.

To approach this kind of questions, one needs to restrict to a well defined space of all distributed algorithms and execution contexts. In general this is an impossible requirement. Yet the distributed algorithms community has come up with some settings that are expressive enough to represent interesting cases and limited enough to start quantifying over "all possible" distributed algorithms [11, 40, 1].

In this work we consider the consensus problem in the Heard-Of model [11]. *Consensus problem* is a central problem in the field of distributed algorithms; it requires that all correct processes eventually decide on one of the initial values. *The Heard-Of model* is a round- and message-passing-based model. It can represent many intricacies of various execution models and yet is simple enough to attempt to analyze it algorithmically [9, 14, 15, 28, 27]. Initially, our goal was to continue the quest from [28] of examining what is algorithmically possible to verify in the Heard-Of model. While working on this problem we have realized that a much more ambitious goal can be achieved: to give a simple, and in particular decidable, characterization of all consensus algorithms in well-defined fragments of the Heard-Of model.

The Heard-Of model is an open ended model: it does not specify what operations processes can perform and what kinds of communication predicates are allowed. Communication predicates in the Heard-Of model capture in an elegant way both synchrony degree and failure model. In this work we fix the set of atomic communication predicates and atomic operations. We opted for a set sufficient to express most prominent consensus algorithms (cf. Section 7), but we do not cover all operations found in the literature on the Heard-Of model.

Our characterization of algorithms that solve consensus is expressed in terms of syntactic conditions both on the text of the algorithm, and on the constraints in the communication predicate. It exhibits an interesting way all consensus algorithms should behave. One could imagine that there can be a consensus algorithm that makes processes gradually converge to a consensus: more and more processes adopting the same value. This is not the case. A consensus algorithm, in models we study here, should have a fixed number of crucial rounds where precise things are guaranteed to happen. Special rounds have been identified for existing algorithms [33], but not their distribution over different phases. Additionally, here we show that all algorithms should have this structure.

As an application of our characterization we can think of using it as an intermediate step in analysis of more complicated settings than the Heard-Of model. An algorithm in a given setting can be abstracted to an algorithm in the Heard-Of model, and then our characterization can be applied. Instead of proving the original algorithm correct it is enough to show that the abstraction is sound. For example, an approach reducing asynchronous semantics to round based semantics under some conditions is developed in [8]. A recent paper [13] gives a reduction methodology in a much larger context, and shows its applicability. The goal language of the reduction is an extension of the Heard-Of model that is not covered by our characterization. As another application, our characterization can be used to quickly see if an algorithm can be improved by taking a less constrained communication predicate, by adapting threshold constants, or by removing parts of code (c.f. Section 7).

Organization of the paper. In the next section we introduce the Heard-Of model and formulate the consensus problem. In the four consecutive sections we present the characterizations for the core model as well as for the extensions with timestamps, coordinators, and with both timestamps and coordinators at the same time. We then give examples of algorithms that are covered by these characterizations. Proofs can be found in the appendix, as well as in the full version of the paper [3].

Related work

The celebrated FLP result [18] states that consensus is impossible to achieve in an asynchronous system in presence of failures, even in the presence of one crash failure. There is a considerable literature investigating the models in which the consensus problem is solvable. Even closer in spirit to the present paper are results on weakest failure detectors required to solve the problem [6, 19]. Another step closer are works providing generic consensus algorithms that can be instantiated to give several known concrete algorithms [31, 22, 21, 5, 34, 33]. The present paper considers a relatively simple model, but gives a characterization result of all possible consensus algorithms.

The cornerstone idea of the Heard-Of model is to represent both asynchrony and failures by the constraints on the message loss expressed by communication predicates. This greatly simplifies the model, that in turn is very useful for a kind of characterizations we present here. Unavoidably, not all aspects of partial synchrony [17, 12] or failures [7] are covered by the model. For example, after a crash it may be difficult for a process to get into initial state, or in terms of the Heard-of model, do the same round as other processes [38, 8]. Faults are not malicious: a sent value may be lost, but the value may not be modified during transmission. These observations just underline that there is no universal model for distributed algorithms. There exists several other proposals of relatively simple and expressible models [20, 40, 1, 32]. The Heard-Of model, while not perfect, is in our opinion representative enough to merit a more detailed study.

On the verification side there are at least three approaches to analysis of the Heard-Of or similar models. One is to use automatic theorem provers like Isabelle [10, 9, 14]. Another is deductive verification methods applied to annotated programs [16, 15]. The closest to this work is a model-checking approach [37, 28, 27, 2]. Particularly relevant here is the work of Maric et al. [28]. who show cut-off results for a fragment of the Heard-Of model and then perform verification on a resulting finite state system. Our fragment of the Heard-Of model is incomparable with the one from that work, and arguably it has less restrictions coming from purely technical issues in proofs. While trying to extend the scope of automatic verification methods along the lines in the above papers, we have realized that in our case it is possible to obtain a characterization result.

Of course there are also other models of distributed systems that are considered in the context of verification. For example there has been big progress on verification of threshold automata [26, 24, 25, 35, 4]. There are also other methods, as automatically generating invariants for distributed algorithms [23, 39, 36], or verification in Coq proof assistant [41, 42].

2 Heard-Of model and the consensus problem

In the Heard-Of model a certain number of processes execute the same code synchronously. An algorithm consists of a sequence of *rounds*, every process executes the same round at the same time. The sequence of rounds, called *phase*, is repeated forever. In a round every process sends the value of one of its variables to a communication medium, receives a multiset of values, and uses it to adopt a new value (cf. Figure 1).

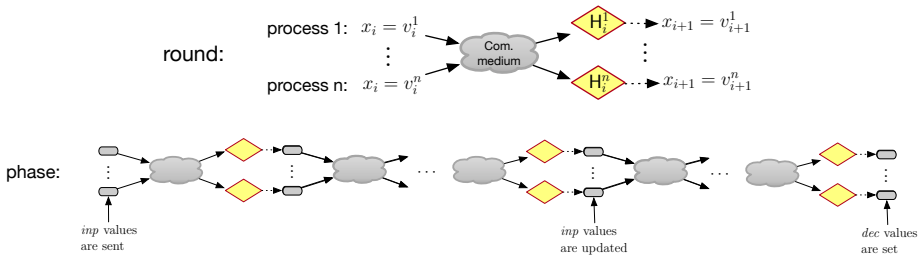

Figure 1 A schema of an execution of a round and of a phase. In every round i every process sends a value of its variable x_i, and sets its variable x_{i+1} depending on the received multiset of values: H_i^j. At the beginning of the phase the value of *inp* is sent, at some round *inp* may be updated; we use **ir** for the index of this round. In the last round *dec* may be set. Both *inp* and *dec* are not updated if the value is ?, standing for undefined.

At the beginning every process has its initial value in variable *inp*. Every process is expected to eventually set its decision variable *dec*. Every round is communication closed meaning that a value sent in a round can only be received in the same round; if it is not received it is lost. A *communication predicate* is used to express a constraint on acceptable message losses. Algorithm 1 is a concrete simple example of a 2-round algorithm with two rounds. In the first round the value of *inp* is send, in the second the value of x_1. We will explain the algorithm later in the text.

We proceed with a description of the syntax and semantics of Heard-Of algorithms. Next we define the consensus problem. In later sections we will extend the core language with timestamps and coordinators.

Algorithm 1 Parametrized OneThird algorithm [11], thr_1, thr_2 are constants from $(0, 1)$.

```
send (inp)
    if uni(H) ∧ |H| > thr_1 · |Π| then x_1 := inp := smor(H);
    if mult(H) ∧ |H| > thr_1 · |Π| then x_1 := inp := smor(H);
send x_1
    if uni(H) ∧ |H| > thr_2 · |Π| then dec := smor(H);
Communication predicate: F(ψ^1 ∧ Fψ^2)
where: ψ^1 := (φ_= ∧ φ_thr1, true)   and   ψ^2 := (φ_thr1, φ_thr2)
```

Syntax

An algorithm has one *phase* that consists of two or more rounds. In the first round each process sends the value of *inp* variable, in the last round it can set the value of *dec* variable. A phase is repeated forever, all processes execute the same round at the same time. A round i is a send statement followed by a sequence of conditionals:

```
send x_{i-1}
    if cond_i^1(H) then x_i := op_i^1(H);
    ⋮
    if cond_i^l(H) then x_i := op_i^l(H);
```

The variables are used in a sequence: first x_0, which is *inp*, is sent and x_1 is set, then x_1 is sent and x_2 is set, etc. (cf. Figure 1). There should be exactly one round (before the last round) where *inp* is updated; the conditional lines in this round are:

$$\texttt{if } cond^j_{\mathbf{ir}}(\mathsf{H}) \texttt{ then } x_{\mathbf{ir}} := inp := \mathtt{op}^j_{\mathbf{ir}}(\mathsf{H})$$

Since this is a special round, we use the index **ir** to designate this round number. In the last round, only instructions setting variable *dec* can be present:

$$\texttt{if } cond^j_r(\mathsf{H}) \texttt{ then } dec := \mathtt{op}^j_r(\mathsf{H})$$

Because of this special form of the last round, a phase needs to have at least two rounds. Of course one can also have a syntax and a characterization for one round algorithms, but unifying the two hinders readability. Our fragment roughly corresponds to the fragment from [28], without extra restrictions but with a less liberty at the fork point.

The intuition behind the syntax is that in the i^{th} round, after a process sends its value of the x_{i-1} variable and receives a multiset H, it finds the first instruction whose condition is satisfied by H and performs the corresponding assignment. Hence, even if multiple conditions are satisfied by a multi-set, only the first such condition is executed.

As an example, consider Algorithm 1. It has two rounds, each begins with a `send` statement. In the first round both x_1 and *inp* are set, in the second round *dec* is set. The conditions talk about properties of the received H multiset; we describe them below.

In round i every process first sends the value of variable x_{i-1}, and then receives a multiset of values H that it uses to set the value of the variable x_i. The possible tests on the received set H are `uni`, `mult`, and $|\mathsf{H}| > thr \cdot |\Pi|$ saying respectively that: the multiset has only one value; has more than one value; and that is of size $> thr \cdot n$ where n is the number of processes and $0 \leq thr < 1$. The possible operations are $\min(\mathsf{H})$ resulting in the minimal value in H, and $\mathrm{smor}(\mathsf{H})$ resulting in the minimal most frequent value in H. For example, the first conditional line in Algorithm 1 tests if there is only one value in H, and if this value has

multiplicity at least $thr_1 \cdot n$ in H; if so *inp* and x_1 are set to this value, it does not matter if min or smor operation is used in this case. The test in the second line holds when received H set has at least two values and is of size at least $thr_1 \cdot n$. In this case x_1 is set to the smallest most frequent value in H.

In addition to description of rounds, an algorithm has also a communication predicate putting constraints on the behavior of the communication medium. A *communication predicate for a phase* with r rounds is a tuple $\psi = (\psi_1, \ldots, \psi_r)$, where each ψ_l is a conjunction of atomic communication predicates that we specify later. A *communication predicate for an algorithm* is

$$(\mathsf{G}\overline{\psi}) \wedge (\mathsf{F}(\psi^1 \wedge \mathsf{F}(\psi^2 \wedge \ldots (\mathsf{F}\psi^k) \ldots)))$$

where $\overline{\psi}$ and ψ^i are communication predicates for a phase. Predicate $\overline{\psi}$ is a *global predicate*, and $\psi^1 \ldots, \psi^k$ are *sporadic predicates*. So the global predicate specifies constraints on every phase of execution, while sporadic predicates specify a *sequence* of special phases that should happen: first ψ_1, followed later by ψ_2, etc. We have two types of atomic communication predicates: $\varphi_=$ says that every process receives the same multiset; φ_{thr} says that every process receives a multiset of size at least $thr \cdot n$ where n is the number of processes. In Algorithm 1 the global predicate is trivial, and we require two special phases. In the first of them, in its first round every process should receive exactly the same H multiset, and the multiset should contain values from at least thr_1 fraction of all processes.

Semantics

The values of variables come from a fixed linearly ordered set D. Additionally, we take a special value $? \notin D$ standing for undefined. We write $D_?$ for $D \cup \{?\}$.

We describe the semantics of an algorithm for n processes. A *state of an algorithm* is a pair of n-tuples of values; denoted (f, d). Intuitively, f specifies the value of the *inp* variable for each process, and d specifies the value of the *dec* variable. The value of *inp* can never be ?, while initially the value of *dec* is ? for every process. We denote by $mset(f)$ the multiset of values appearing in the tuple f. Only values of *inp* and *dec* survive between phases. All the other variables are reset to ? at the beginning of each phase.

There are two kinds of transitions:

$(f, d) \xrightarrow{\psi} (f', d')$	a phase transition
$f \xRightarrow{\varphi}_i f'$	a transition for round i

Phase transitions will be defined as a composition of round transitions. In a transition for round i, tuple f describes the values of x_{i-1}, and f' the values of x_i Phase transition is labeled with a phase communication predicate, while a round transition has a round number and a conjunction of atomic predicates as labels.

Before defining these transitions we need to describe the semantics of communication predicates. At every round processes send values of their variable to a communication medium, and then receive a multiset of values from the medium (cf. Figure 1). Communication medium is not assumed to be perfect, it can send a different multiset of values to every process, provided it is a sub-multiset of received values. An atomic communication predicate puts constraints on multisets that every process receives. In other words, such a predicate specifies constraints on a tuple of multisets $\vec{\mathsf{H}} = (\mathsf{H}_1, \ldots, \mathsf{H}_n)$. Predicate $\varphi_=$ requires that all the multisets are the same. Predicate φ_{thr} requires that every multiset is bigger than $thr \cdot n$ for some number $0 \leq thr < 1$. Predicate *true* does not put any restrictions. We write $\vec{\mathsf{H}} \vDash \varphi$ when the tuple of multisets $\vec{\mathsf{H}}$ satisfies the conjunction of atomic predicates φ.

Once a process p receives a multiset H_p, it uses it to do an update of one of its variables. For this it finds the first condition that H_p satisfies and performs the operation from the corresponding assignment.

Recall that a *condition* is a conjunction of atomic conditions: `uni`, `mult`, $|\mathsf{H}| > thr \cdot |\Pi|$. A multiset H satisfies `uni` when it contains just one value; it satisfies `mult` if it contains more than one value. A multiset H satisfies $|\mathsf{H}| > thr \cdot |\Pi|$ when the size of H is bigger than $thr \cdot n$, where n is the number of processes. Observe that only predicates of the last type take into account possible repetitions of the same value.

We can now define the *update value* $\texttt{update}_i(\mathsf{H})$, describing to which value the process sets its variable in round i upon receiving the multiset H. For this the process finds the first conditional statement in the sequence of instructions for round i whose condition is satisfied by $\mathsf{H} - \{?\}$ and looks at the operation in the statement:

- if it is $x := \min(\mathsf{H})$ then $\texttt{update}_i(\mathsf{H})$ is the minimal value in $\mathsf{H} - \{?\}$;
- if it is $x := \mathrm{smor}(\mathsf{H})$ then $\texttt{update}_i(\mathsf{H})$ is the smallest most frequent value in $\mathsf{H} - \{?\}$;
- if no condition is satisfied then $\texttt{update}_i(\mathsf{H}) = ?$.

A transition $f \overset{\varphi}{\Longrightarrow}_i f'$ is possible when there exists a tuple of multisets $(\mathsf{H}_1, \ldots, \mathsf{H}_n) \vDash \varphi$ such that for all $p = 1, \ldots, n$: $\mathsf{H}_p \subseteq mset(f)$, and $f'(p) = \texttt{update}_i(\mathsf{H}_p)$. Observe that ? value in H_p is ignored by the `update` function, but not by the communication predicate.

A transition $(f, d) \xrightarrow{\psi} (f', d')$, for $\psi = (\varphi_1, \ldots, \varphi_n)$, is possible when there is a sequence:

$$f_0 \overset{\varphi_1}{\Longrightarrow}_1 f_1 \overset{\varphi_2}{\Longrightarrow}_2 \cdots \overset{\varphi_{r-1}}{\Longrightarrow}_{r-1} f_{r-1} \overset{\varphi_r}{\Longrightarrow}_r f_r \qquad \text{where}$$

- $f_0 = f$;
- $f'(p) = f_{\mathbf{ir}}(p)$ if $f_{\mathbf{ir}}(p) \neq ?$, and $f'(p) = f_{\mathbf{ir}}(p)$ otherwise;
- $d'(p) = d(p)$ if $d(p) \neq ?$, and $d'(p) = f_r(p)$ otherwise.

This means that if in round $\mathbf{ir}$, the value of $f_{\mathbf{ir}}(p)$ was ?, then the process p retains its value of the *inp* onto the next phase; otherwise the process p updates its value of *inp* to $f_{\mathbf{ir}}(p)$. The value of *dec* cannot be updated, it can only be set if it has not been set before. For setting the value of *dec*, the value from the last round is used.

An *execution* is a sequence of phase transitions. An *execution of an algorithm respecting a communication predicate* $(\mathsf{G}\overline{\psi}) \wedge (\mathsf{F}(\psi^1 \wedge \mathsf{F}(\psi^2 \wedge \ldots (\mathsf{F}\psi^k) \ldots)))$ is an infinite sequence:

$$(f_0, d_0) \xrightarrow{\overline{\psi}}{}^{*} (f_1, d_1) \xrightarrow{\psi \wedge \psi^1} (f'_1, d'_1) \cdots \xrightarrow{\overline{\psi}}{}^{*} (f_k, d_k) \xrightarrow{\psi \wedge \psi^k} (f'_k, d'_k) \xrightarrow{\overline{\psi}}{}^{\omega} \cdots$$

where $\xrightarrow{\overline{\psi}}{}^{*}$ stands for a finite sequence of $\xrightarrow{\overline{\psi}}$ transitions, and $\xrightarrow{\overline{\psi}}{}^{\omega}$ for an infinite sequence. For every execution there is some fixed n determining the number of processes, f_0 is any n-tuple of values without ?, and d_0 is the n-tuple of ? values. Observe that the size of the first tuple determines the size of every other tuple. By definition of transitions, there is always a transition from every configuration, so an execution cannot block. Thus we can think of every execution as being infinite.

▶ **Definition 1** (Consensus problem). *An algorithm has* agreement property *if for every number of processes n, and for every state (f, d) reachable by an execution of the algorithm, for all processes p_1 and p_2, either $d(p_1) = d(p_2)$ or one of the two values is ?. An algorithm has* termination property *if for every n, and for every execution there is a state (f, d) on this execution with $d(p) \neq ?$ for all $p = 1, \ldots, n$. An algorithm* solves consensus *if it has agreement and termination properties.*

▶ Remark 2. Normally, the consensus problem also requires irrevocability and integrity properties, but these are always guaranteed by the semantics: once set, a process cannot change its *dec* value, and a variable can be set only to one of the values that has been received.

▶ Remark 3. The original definition of the Heard-Of model is open ended: it does not limit possible forms of a communication predicate, conditions, or operations. Clearly, for the kind of result we present here, we need to fix them.

▶ Remark 4. In the original definition processes are allowed to have identifiers. We do not need them for the set of operations we consider. Later we will add coordinators without referring to identifiers. This is a relatively standard way of avoiding identifiers while having reasonable expressivity.

3 A characterization for the core language

We present a characterization of all the algorithms in our language that solve consensus. In later sections we will extend it to include timestamps and coordinators. As it will turn out, for our analysis we will need to consider only two values a, b with a fixed order between them: we take a smaller than b. This order influences the semantics of instructions: the result of min is a on a multiset containing at least one a; the result of smor is a on a multiset with the same number of a's and b's. Because of this asymmetry we mostly focus on the number of b's in a tuple. In our analysis we will consider tuples of the form $bias(\theta)$ for $\theta < 1$, i.e., a tuple where we have n processes (for some large enough n), out of which $\theta \cdot n$ of them have their value set to b; and the remaining ones to a. The tuple containing only b's (resp. only a's) is called $solo$ (resp. $solo^a$).

We show that there is essentially one way to solve consensus. The text of the algorithm together with the form of the global predicate determines a threshold $\overline{thr}$. We prove that in the language we consider here, there should be a *unifier phase* which guarantees that the tuple of *inp* values after the phase belongs to one of the following four types: $solo$, $solo^a$, $bias(\theta)$, or $bias(1-\theta)$ where $\theta \geq \overline{thr}$. Intuitively, this means that there is a dominant value in the tuple. This phase should be followed by a *decider phase* which guarantees that if the tuple of *inp* values is of one of the above mentioned types, then all the processes decide. While this ensures termination, agreement is ensured by proving that some structural properties on the algorithm should always hold.

Before stating the characterization, we will make some observations that allow us to simplify the structure of an algorithm, and in consequence simplify the statements.

It is easy to see that in our languge we can assume that the list of conditional instructions in each round can have at most one `uni` conditional followed by a sequence of `mult` conditionals with non-increasing thresholds:

$$\texttt{if uni}(\mathsf{H}) \wedge |\mathsf{H}| > thr^i_u \cdot |\Pi| \texttt{ then } x := \texttt{op}^i_u(\mathsf{H})$$
$$\texttt{if mult}(\mathsf{H}) \wedge |\mathsf{H}| > thr^{i,1}_m \cdot |\Pi| \texttt{ then } x := \texttt{op}^i_m(\mathsf{H})$$
$$\vdots$$
$$\texttt{if mult}(\mathsf{H}) \wedge |\mathsf{H}| > thr^{i,k}_m \cdot |\Pi| \texttt{ then } x := \texttt{op}^i_m(\mathsf{H})$$

We use superscript i to denote the round number: so thr^1_u is a threshold associated to `uni` instruction in the first round, etc. If round i does not have a `uni` instruction, then thr^1_u will be -1. For the sake of brevity, $thr^{i,k}_m$ will always denote the minimal threshold appearing in any of the `mult` instructions in round i and -1 if no `mult` instructions exist in round i.

We fix a *communication predicate*:

$$(\mathsf{G}\overline{\psi}) \wedge (\mathsf{F}(\psi^1 \wedge \mathsf{F}(\psi^2 \wedge \ldots (\mathsf{F}\psi^k)\ldots))) \tag{1}$$

Without loss of generality we can assume that every sporadic predicate implies the global predicate; in consequence, $\overline{\psi} \wedge \psi^i$ is equivalent to ψ^i. Recall that each of $\overline{\psi}, \psi^1, \ldots, \psi^k$ is an r-tuple of conjunctions of atomic predicates. We write $\psi\lfloor_i$ for the i-th element of the tuple and so ψ is $(\psi\lfloor_1, \ldots, \psi\lfloor_r)$. By $thr_i(\psi)$ we denote the threshold constant appearing in the predicate $\psi\lfloor_i$, i.e., if $\psi\lfloor_i$ has φ_{thr} as a conjunct, then $thr_i(\psi) = thr$, if it has no such conjunct then $thr_i(\psi) = -1$. We call $\psi\lfloor_i$ an *equalizer* if it has $\varphi_=$ as a conjunct. In this case we also say that ψ has an equalizer.

Recall (cf. page 2) that a transition $f \overset{\psi}{\Longrightarrow}_i f'$ for a round i under a phase predicate ψ is possible when there is a tuple of multisets $(\mathsf{H}_1, \ldots, \mathsf{H}_n) \vDash \psi\lfloor_i$ such that for all $p = 1, \ldots, n$: $\mathsf{H}_p \in mset(f)$ and $f'(p) = \mathtt{update}_i(\mathsf{H}_p)$.

▶ **Definition 5.** *A round i is* preserving *w.r.t. ψ iff one of the three conditions hold: (i) it does not have an* `uni` *instruction, (ii) it does not have a* `mult` *instruction, or (iii) $thr_i(\psi) < \max(thr^i_u, thr^{i,k}_m)$. Otherwise the round is* non-preserving. *The round is* solo safe *w.r.t. ψ if $0 \leq thr^i_u \leq thr_i(\psi)$.*

If i is a preserving round, then there exists a tuple f such that $? \notin mset(f)$ and such that a transition $f \overset{\psi}{\Longrightarrow}_i f'$ is possible for f' a tuple consisting solely of ?. The consequence of such a transition is that *inp* is not updated in the phase, i.e., old values of *inp* are *preserved*. On the other extreme, if all transitions in the phase are non-preserving then all *inp* values are necessarily updated by the phase. Finally, a solo safe round cannot alter the *solo* state, i.e., *solo* $\overset{\psi}{\Longrightarrow}_i$ *solo* is the only transition possible from *solo*.

▶ Remark 6. Suppose rounds $1, \ldots, i-1$ are non-preserving under $\overline{\psi}$, the global predicate. In this situation, since $? \notin mset(f)$, if $f \overset{\overline{\psi}\lfloor_1}{\Longrightarrow}_1 f_1 \overset{\overline{\psi}\lfloor_2}{\Longrightarrow}_2 \ldots \overset{\overline{\psi}\lfloor_{i-1}}{\Longrightarrow}_{i-1} f_{i-1}$ then $? \notin mset(f_{i-1})$. Hence, no heard-of multi-set H constructed from f_{i-1} can have ? value. Notice that every process is bound to receive a heard-of set of size at least $thr_i(\overline{\psi})$ in round i. For a sake of example, suppose $thr_i(\overline{\psi}) > thr^{i,2}_m$. The semantics then guarantees that every heard-of set sent during the i^{th} round either satisfies the `uni` instruction, or one of the first two `mult` instructions, or no instruction at all. Hence, in such a case all the `mult` instructions except the first two can be removed from the description of round i as they will be never executed. This implies that we can adopt the following assumption.

▶ **Assumption 1.** *For every round i, if rounds $1, \ldots, i-1$ are non-preserving under $\overline{\psi}$ then*

$$\begin{cases} thr^i_u \geq thr_i(\overline{\psi}) & \text{if round } i \text{ has } \mathtt{uni} \text{ instruction} \\ thr^{i,k}_m \geq thr_i(\overline{\psi}) & \text{if round } i \text{ has } \mathtt{mult} \text{ instruction} \end{cases} \tag{2}$$

We put some restrictions on the form of algorithms we consider in our characterization. They greatly simplify the statements, and as we argue, are removing cases that are not that interesting anyway.

▶ **Proviso 1.** *We adopt the following additional syntactic restrictions:*

- *We require that the global predicate does not have an equalizer.*
- *We assume that there is no* `mult` *instruction in the round* $\mathbf{ir} + 1$.

Concerning the first of the above requirements, if the global predicate has an equalizer then it is quite easy to construct an algorithm for consensus because equalizer guarantees that in a given round all the processes receive the same value. The characterization below

can be extended to this case but would require to mention it separately in all the statements. Concerning the second requirement, We can show that if such a `mult` instruction exists then either the algorithm violates consensus, or the instruction will never be fired in any execution of the algorithm and so it can be removed without making an algorithm incorrect.

In order to state our characterization we need to give formal definitions of concepts we have discussed at the beginning of the section.

▶ **Definition 7.** *The* border threshold *is* $\overline{thr} = \max(1 - thr_u^1, 1 - thr_m^{1,k}/2)$.

▶ **Definition 8.** *A predicate* ψ *is a*

- Decider*, if all rounds are solo safe w.r.t.* ψ
- Unifier*, if the three conditions hold:*
  - $thr_1(\psi) \geq thr_m^{1,k}$ *and either* $thr_1(\psi) \geq thr_u^1$ *or* $thr_1(\psi) \geq \overline{thr}$*,*
  - *there exists* i *such that* $1 \leq i \leq \mathbf{ir}$ *and* $\psi|_i$ *is an equalizer,*
  - *rounds* $2, \ldots, i$ *are non-preserving w.r.t.* ψ *and rounds* $i+1, \ldots \mathbf{ir}$ *are solo-safe w.r.t.* ψ

Finally, we list some syntactic properties of algorithms that, as we will see later, imply the agreement property.

▶ **Definition 9.** *An algorithm is* syntactically safe *when:*

1. *First round has a* `mult` *instruction.*
2. *Every round has a* `uni` *instruction.*
3. *In the first round the operation in every* `mult` *instruction is* smor*.*
4. $thr_m^{1,k}/2 \geq 1 - thr_u^{\mathbf{ir}+1}$*, and* $thr_u^1 \geq 1 - thr_u^{\mathbf{ir}+1}$*.*

Recall that $\psi^1, \ldots, \psi^k$ are the set of sporadic predicates from the communication predicate. Without loss of generality we can assume that there is at least one sporadic predicate; at a degenerate case it is always possible to take a sporadic predicate that is the same as the global predicate. With these definitions we can state our characterization:

▶ **Theorem 10.** *Consider algorithms in the core language satisfying syntactic constraints from Assumption 1 and Proviso 1. An algorithm solves consensus iff it is syntactically safe according to Definition 9, and it satisfies the condition:*

T *There is* $i \leq j$ *such that* ψ^i *is a unifier and* ψ^j *is a decider.*

A two value principle is a corollary from the proof of the above theorem: an algorithm solves consensus iff it solves consensus for two values. Indeed, it turns out that it is enough to work with three values a, b, and ? standing for undefined. The proof considers separately safety and liveness aspects of the consensus problem. Notice that the properties from Definition 9 intervene also in the proof of termination.

▶ **Lemma 11.** *An algorithm violating structural properties from Definition 9 cannot solve consensus. An algorithm with the structural properties has the agreement property.*

▶ **Lemma 12.** *An algorithm with the structural properties from Definition 9 has the termination property iff it satisfies condition T from Theorem 10.*

4 A characterization for algorithms with timestamps

We extend our characterization to algorithms with timestamps. Now, variable *inp* stores not only the value but also a timestamp, that is the number of the last phase at which *inp* was updated. These timestamps are used in the first round, as a process considers only values

with the most recent timestamp. The syntax is the same as before except that we introduce a new operation, called maxts, that must be used in the first round and nowhere else. So the form of the first round becomes:

```
send (inp, ts)
    if cond_1^1(H) then x_1 := maxts(H);
    ⋮
    if cond_1^l(H) then x_1 := maxts(H);
```

The semantics of transitions for rounds and phases needs to take into account timestamps. The semantics changes only for the first round; its form becomes $(f,t) \xRightarrow{\varphi} f'$, where t is a vector of timestamps (n-tuple of natural numbers). Timestamps are ignored by communication predicates and conditions, but are used in the update operation. The operation maxts(H) returns the smallest among values with the most recent timestamp in H.

The form of a phase transition changes to $(f,t,d) \xrightarrow{\psi} (f',t',d')$. Value $t(p)$ is the timestamp of the last update of *inp* of process p (whose value is $f(p)$). We do not need to keep timestamps for d since the value of *dec* can be set only once. Phase transitions are defined as before, taking into account the above mentioned change for the first round transition, and the fact that in the round **ir** when *inp* is updated then so is its timestamp. Some examples of algorithms with timestamps are presented in Section 7.

As in the case of the core language, without loss of generality we can assume conditions from Assumption 1. Concerning Proviso 1, we assume almost the same conditions, but now the second one refers to the round **ir** and not to the round **ir** + 1, and is a bit stronger.

▶ **Proviso 2.** *We adopt the following syntactic restrictions:*

- *We require that the global predicate does not have an equalizer.*
- *We assume that there is no* `mult` *instruction in the round* **ir**, *and that* $thr_u^{\mathbf{ir}} \geq 1/2$.

The justification for the first restriction is as before. Concerning the second restriction, we can prove that if these two assumptions do not hold then either the algorithm violates consensus, or we can remove the `mult` instruction and increase $thr_u^{\mathbf{ir}}$ without making an algorithm incorrect.

Our characterization resembles the one for the core language. The structural conditions get slightly modified: the condition on constants is weakened, and there is no need to talk about smor operations in the fist round.

▶ **Definition 13.** *An algorithm is* syntactically t-safe *when:*

1. *Every round has a* `uni` *instruction.*
2. *First round has a* `mult` *instruction.*
3. $thr_m^{1,k} \geq 1 - thr_u^{\mathbf{ir}+1}$ *and* $thr_u^1 \geq 1 - thr_u^{\mathbf{ir}+1}$.

We consider the same shape of a communication predicate as in the case of the core language (1). A characterization for the case with timestamps uses a stronger version of a unifier that we define now. The intuition is that we do not have $\overline{thr}$ constant because of maxts operations in the first round. In other words, the conditions are the same as before but when taking $\overline{thr} > 1$.

▶ **Definition 14.** *A predicate ψ is a* strong unifier *ψ if it is a unifier in a sense of Definition 8 and $thr_u^1 \leq thr_1(\psi)$.*

Modulo the above two changes, the characterization stays the same.

▶ **Theorem 15.** *Consider algorithms in the language with timestamps satisfying syntactic constraints from Assumption 1 and Proviso 2. An algorithm satisfies consensus iff it is syntactically t-safe according to Definition 13, and it satisfies:*
sT *There are $i \leq j$ such that ψ^i is a strong unifier and ψ^j is a decider.*

5 A characterization for algorithms with coordinators

We consider algorithms equipped with coordinators. The novelty is that we can now have rounds where there is a unique process that receives values from other processes, as well as rounds where there is a unique process that sends values to other processes. For this we extend the syntax by introducing a round type that can be: `every`, `lr` (leader receive), or `ls` (leader-send):

- A round of type `every` behaves as before.
- In a round of type `lr` only one arbitrarily selected process receives values.
- In a round of type `ls`, the process selected in the immediately preceding `lr` round sends its value to all other processes.

If an `ls` round is not preceded by an `lr` round then an arbitrarily chosen process sends its value. We assume that every `lr` round is immediately followed by an `ls` round, because otherwise the `lr` round would be useless. We also assume that *inp* and *dec* are not updated during `lr` rounds, as only one process is active in these rounds.

For `ls` rounds we introduce a new communication predicate. The predicate $\varphi_{\texttt{ls}}$ says that the leader successfully sends its message to everybody; it makes sense only for `ls` rounds.

These extensions of the syntax are reflected in the semantics. For convenience we introduce two new names for tuples: one^b is a tuple where all the entries are ? except for one entry which is b; similarly for one^a. Abusing the notation we also write $one^?$ for $solo^?$, namely the tuple consisting only of ? values.

Let us define the semantics of `lr` and `ls` rounds. If i-th round is of type `lr`, we have a transition $f \overset{\psi}{\Longrightarrow}_i one^d$ for every $d \in \mathrm{fire}_i(f, \psi)$. In particular, if $? \in \mathrm{fire}_i(f, \psi)$ then $f \overset{\varphi}{\Longrightarrow}_i solo^?$ is possible.

Suppose i-th round is of type `ls`. If $\psi|_i$ contains $\varphi_{\texttt{ls}}$ as a conjunct then

$$one^d \overset{\psi}{\Longrightarrow}_i solo^d \qquad \text{if round } (i-1) \text{ is of type } \texttt{lr}$$
$$f \overset{\psi}{\Longrightarrow}_i solo^d \quad \text{for } d \in set(f) \qquad \text{otherwise}$$

When $\psi|_i$ does not contain $\varphi_{\texttt{ls}}$ then independently of the type of the round $(i-1)$ we have $f \overset{\psi}{\Longrightarrow}_i f'$ for every $d \in set(f)$ and f' such that $set(f') \subseteq \{d, ?\}$.

We consider the same shape of a communication predicate as in the case of the core language (1).

The semantics allows us to adopt some more simplifying assumptions about the syntax of the algorithm, and the form of the communication predicate.

▶ **Assumption 2.** *We assume that* `ls` *rounds do not have a* `mult` *instruction. Indeed, from the above semantics it follows that* `mult` *instruction is never used in a round of type* `ls`*. It also does not make much sense to use $\varphi_{\texttt{ls}}$ in rounds other than of type* `ls`*. So to shorten some definitions we require that $\varphi_{\texttt{ls}}$ can appear only in communication predicates for* `ls`*-rounds. For similar reasons we require that $\varphi_=$ predicate is not used in* `ls`*-rounds. As we have observed in the first paragraph, we can assume that neither round* **ir** *nor the last round are of type* `lr`*.*

The notions of preserving and solo-safe rounds get adapted to the new syntax.

▶ **Definition 16.** *A round of type* `ls` *is* c-solo-safe *w.r.t.* ψ *if* ψ_i *has* $\varphi_{\texttt{ls}}$ *as a conjunct, it is* c-preserving *otherwise. A round of type other than* `ls` *is* c-preserving *or* c-solo-safe *w.r.t* ψ *if it is so in the sense of Definition 5.*

▶ **Definition 17.** *A* c-equalizer *is a conjunction containing a term of the form* $\varphi_=$ *or* $\varphi_{\texttt{ls}}$.

▶ **Proviso 3.** *We assume the same conditions as in Proviso 8, but using the concepts of c-equalizers instead of equalizers.*

To justify the proviso we prove that `mult` instruction in round $\mathbf{ir}+1$ cannot be useful. Assumption 1 is also updated to using the notion of c-preserving instead of preserving. We restate it for convenience.

▶ **Assumption 3.** *For every round* i*, if rounds* $1,\ldots,i-1$ *are non-c-preserving under* $\overline{\psi}$ *then*

$$\begin{cases} thr_u^i \geq thr_i(\overline{\psi}) & \text{if round } i \text{ has } \texttt{uni} \text{ instruction} \\ thr_m^{i,k} \geq thr_i(\overline{\psi}) & \text{if round } i \text{ has } \texttt{mult} \text{ instruction} \end{cases} \tag{3}$$

Finally, the above modifications imply modifications of terms from Definition 8.

▶ **Definition 18.** *A predicate* ψ *is called a*

- c-decider*, if all rounds are c-solo safe w.r.t.* ψ*.*
- c-unifier*, if*
  - $thr_1(\psi) \geq thr_m^{1,k}$ *and either* $thr_1(\psi) \geq thr_u^1$ *or* $thr_1(\psi) \geq \overline{thr}$*,*
  - *there exists* i *such that* $1 \leq i \leq \mathbf{ir}$ *and* $\psi|_i$ *is an c-equalizer,*
  - *rounds* $2,\ldots,i$ *are non-c-preserving w.r.t.* ψ *and rounds* $i+1,\ldots\mathbf{ir}$ *are c-solo-safe w.r.t.* ψ*.*

With these modifications, we get an analog of Theorem 10 for the case with coordinators subject to the modified provisos as explained above.

▶ **Theorem 19.** *Consider algorithms in the language with timestamps satisfying syntactic constraints from Assumptions 2, 3 and Proviso 3. An algorithm satisfies consensus iff the first round and the* $(\mathbf{ir}+1)^{th}$ *round are not of type* `ls`*, it is syntactically safe according to Definition 9, and it satisfies the condition:*
cT *There are* $i \leq j$ *such that* ψ^i *is a c-unifier and* ψ^j *is a c-decider.*

6 A characterization for algorithms with coordinators and timestamps

Finally, we consider an extension of the core language with both coordinators and timestamps. Formally, we extend the coordinator model with timestamps in the same way we have extended the core model. So now *inp* variables store pairs (value, timestamp), and all the instructions in the first round are maxts (cf. page 10).

▶ **Proviso 4.** *We assume the same proviso as for timestamps; namely, Proviso 2, but using the notion of c-equalizer.*

As in the previous cases we justify our proviso by showing that the algorithm violating the second condition would not be correct or the condition could be removed.

The characterization is a mix of conditions from timestamps and coordinator cases.

▶ **Definition 20.** *A predicate ψ is a strong c-unifier if it is a c-unifier (cf. Definition 14) and $thr_u^1 \leq thr_1(\psi)$.*

▶ **Theorem 21.** *Consider algorithms in the language with timestamps satisfying syntactic constraints from Assumptions 2, 3 and Proviso 4. An algorithm satisfies consensus iff the first round and the $(\mathbf{ir}+1)^{th}$ round are not of type* `ls`*, it has the structural properties from Definition 13, and it satisfies:*

scT *There are $i \leq j$ such that ψ^i is a strong c-unifier and ψ^j is a c-decider.*

7 Examples

We apply the characterizations from the previous sections to some consensus algorithms studied in the literature, and their variants.

First, we can revisit the parametrized Algorithm 1 from page 4. This is an algorithm in the core language, and it depends on two thresholds. Theorem 10 implies that it solves consensus iff $thr_1/2 \geq 1 - thr_2$. In case of $thr_1 = thr_2 = 2/3$ we obtain the well known OneThird algorithm. But, for example, $thr_1 = 1/2$ and $thr_2 = 3/4$ are also possible solutions for this inequality. So Algorithm 1 solves consensus for these values of thresholds.

Because of the conditions on constants, $thr_m^{1,k}/2 \geq 1 - thr_u^{\mathbf{ir}+1}$ coming from Definition 9, it is not possible to have an algorithm in the core language where all constants are at most $1/2$. This answers a question from [11] for the language we consider here.

The above condition on constants is weakened to $thr_m^{1,k} \geq 1 - thr_u^{\mathbf{ir}+1}$ when we have timestamps. In this case indeed it is possible to use only $1/2$ thresholds [27].

When we have both timestamps and coordinators, we get variants of Paxos algorithm.

Algorithm 2 Paxos algorithm.

`send` (inp, ts) `lr`
  `if uni`(H) $\wedge$ |H| $> 1/2 \cdot |\Pi|$ `then` $x_1 :=$ maxts(H);
  `if mult`(H) $\wedge$ |H| $> 1/2 \cdot |\Pi|$ `then` $x_1 :=$ maxts(H);
`send` x_1 `ls`
  `if uni`(H) `then` $x_2 := inp :=$ smor(H);
`send` x_2 `lr`
  `if uni`(H) $\wedge$ |H| $> 1/2 \cdot |\Pi|$ `then` $x_3 :=$ smor(H);
`send` x_3 `ls`
  `if uni`(H) `then` $dec :=$ smor(H);
`Communication predicate:` $\mathsf{F}(\psi^1)$ where $\psi^1 := (\varphi_{1/2},\ \varphi_{\mathtt{ls}},\ \varphi_{1/2},\ \varphi_{\mathtt{ls}})$

The algorithm is correct by Theorem 21. One can observe that without modifying the code there is not much room for improvement in this algorithm. A decider phase is needed to solve consensus, and ψ_1 is a minimal requirement for a decider phase. A possible modification is to change the thresholds in the first round to, say, $1/3$ and in the third round to $2/3$ (both in the algorithm and in the communication predicate).

Algorithm 3 Three round Paxos algorithm.

`send` (inp, ts) `lr`
  `if` $\mathtt{uni}(\mathsf{H}) \wedge |\mathsf{H}| > 1/2 \cdot |\Pi|$ `then` $x_1 := \mathrm{maxts}(\mathsf{H})$;
  `if` $\mathtt{mult}(\mathsf{H}) \wedge |\mathsf{H}| > 1/2 \cdot |\Pi|$ `then` $x_1 := \mathrm{maxts}(\mathsf{H})$;
`send` x_1 `ls`
  `if` $\mathtt{uni}(\mathsf{H})$ `then` $x_2 := inp := \mathrm{smor}(\mathsf{H})$;
`send` x_2 `every`
  `if` $\mathtt{uni}(\mathsf{H}) \wedge |\mathsf{H}| > 1/2 \cdot |\Pi|$ `then` $dec := \mathrm{smor}(\mathsf{H})$;
`Communication predicate:` $\mathsf{F}(\psi^1)$ where $\psi^1 := (\varphi_{1/2},\ \varphi_{\mathtt{ls}},\ \varphi_{1/2})$

The three round Paxos presented above is also correct by Theorem 21. Once again it is possible to change constants in the first round to 1/3 and in the last round to 2/3 (both in the algorithm and in the communication predicate).

One can also wander about algorithms with coordinators but without timestamps. Here is a possibility that resembles three round Paxos:

Algorithm 4 Three round coordinator algorithm.

`send` (inp) `lr`
  `if` $\mathtt{uni}(\mathsf{H}) \wedge |\mathsf{H}| > 2/3 \cdot |\Pi|$ `then` $x_1 := \mathrm{smor}(\mathsf{H})$;
  `if` $\mathtt{mult}(\mathsf{H}) \wedge |\mathsf{H}| > 2/3 \cdot |\Pi|$ `then` $x_1 := \mathrm{smor}(\mathsf{H})$;
`send` x_1 `ls`
  `if` $\mathtt{uni}(\mathsf{H})$ `then` $x_2 := inp := \mathrm{smor}(\mathsf{H})$;
`send` x_2 `every`
  `if` $\mathtt{uni}(\mathsf{H}) \wedge |\mathsf{H}| > 2/3 \cdot |\Pi|$ `then` $dec := \mathrm{smor}(\mathsf{H})$;
`Communication predicate:` $\mathsf{F}(\psi)$ where $\psi := (\varphi_{2/3}, \varphi_{\mathtt{ls}}, \varphi_{2/3})$

The algorithm solves consensus by Theorem 19. The constants are bigger than in Paxos because we do not have timestamps: the constraints on constants come from Definition 9, and not from Definition 13. The advantage is that we do not need time-stamps, while keeping the same structure as for three-round Paxos.

It is possible to introduce more parameters in these algorithms to analyze for which choices of parameters they solve consensus.

8 Conclusions

We have characterized all algorithms solving consensus in a fragment of the Heard-Of model. We have aimed at a fragment that can express most important algorithms while trying to avoid ad hoc restrictions (c.f. proviso on page 8). The fragment covers algorithms considered in the context of verification [28, 10] with a notable exception of algorithms sending more than one variable. In this work we have considered only single phase algorithms while originally the model permits also to have initial phases. We believe that it is possible to extend the characterization to incorporate the initial phases, but this would further complicate the results and there are no well-know algorithms that use such phases. More severe and technically important restriction is that we allow to use only one variable at a time. In particular, it is not possible to send pairs of variables.

One curious direction of further research would be to list all "best" consensus algorithms under some external constraints; for example the constraints can come from some properties of an execution platform external to the Heard-Of model. This problem assumes that there

is some way to compare two algorithms. One guiding principle for such a measure could be efficient use of knowledge [30, 29]: at every step the algorithm does maximum it can do, given its knowledge of the state of the system.

This research is on the borderline between distributed computing and verification. From a distributed computing side it considers quite a simple model, but gives a characterization result. From a verification side, the systems are complicated because the number of processes is unbounded, there are timestamps, and interactions are based on a fraction of processes having a particular value. We do not advance on verification methods for such a setting. Instead, we observe that in the context considered here verification may be avoided. We believe that a similar phenomenon can appear also for other problems than consensus. It is also an intriguing question to explore how much we can enrich the current model and still get a characterization. We conjecture that a characterization is possible for an extension with randomness covering at least the Ben-Or algorithm. Of course, formalization of proofs, either in Coq or Isabelle, for such extensions would be very helpful.

References

1 Marcos K. Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Partial synchrony based on set timeliness. *Distributed Computing*, 25(3):249–260, 2012. doi:10.1007/s00446-012-0158-8.

2 Benjamin Aminof, Sasha Rubin, Ilina Stoilkovska, Josef Widder, and Florian Zuleger. Parameterized model checking of synchronous distributed algorithms by abstraction. In Isil Dillig and Jens Palsberg, editors, *Verification, Model Checking, and Abstract Interpretation - 19th International Conference, VMCAI 2018*, volume 10747 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2018. doi:10.1007/978-3-319-73721-8_1.

3 A.R. Balasubramanian and Igor Walukiewicz. Characterizing consensus in the heard-of model. arXiv:2004.09621.

4 Nathalie Bertrand, Igor Konnov, Marijana Lazic, and Josef Widder. Verification of randomized consensus algorithms under round-rigid adversaries. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory*, volume 140 of *LIPIcs*, pages 33:1–33:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.33.

5 Martin Biely, Josef Widder, Bernadette Charron-Bost, Antoine Gaillard, Martin Hutle, and André Schiper. Tolerating corrupted communication. In Indranil Gupta and Roger Wattenhofer, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 2007*, pages 244–253. ACM, 2007. doi:10.1145/1281100.1281136.

6 Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, 1996. doi:10.1145/234533.234549.

7 Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996. doi:10.1145/226643.226647.

8 Mouna Chaouch-Saad, Bernadette Charron-Bost, and Stephan Merz. A reduction theorem for the verification of round-based distributed algorithms. In Olivier Bournez and Igor Potapov, editors, *Reachability Problems, 3rd International Workshop, RP 2009*, volume 5797 of *Lecture Notes in Computer Science*, pages 93–106. Springer, 2009. doi:10.1007/978-3-642-04420-5_10.

9 Bernadette Charron-Bost, Henri Debrat, and Stephan Merz. Formal verification of consensus algorithms tolerating malicious faults. In Xavier Défago, Franck Petit, and Vincent Villain, editors, *Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, SSS 2011*, volume 6976 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 2011. doi:10.1007/978-3-642-24550-3_11.

10 Bernadette Charron-Bost and Stephan Merz. Formal verification of a consensus algorithm in the heard-of model. *Int. J. Software and Informatics*, 3(2-3):273–303, 2009. URL: http://www.ijsi.org/ch/reader/view_abstract.aspx?file_no=273&flag=1.

11 Bernadette Charron-Bost and André Schiper. The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, 2009.

12 Flaviu Cristian and Christof Fetzer. The timed asynchronous distributed system model. *IEEE Trans. Parallel Distrib. Syst.*, 10(6):642–657, 1999. doi:10.1109/71.774912.

13 Andrei Damian, Cezara Dragoi, Alexandru Militaru, and Josef Widder. Communication-closed asynchronous protocols. In *CAV (2)*, volume 11562 of *Lecture Notes in Computer Science*, pages 344–363. Springer, 2019. doi:10.1007/978-3-030-25543-5_20.

14 Henri Debrat and Stephan Merz. Verifying fault-tolerant distributed algorithms in the heard-of model. *Archive of Formal Proofs*, 2012, 2012. URL: https://www.isa-afp.org/entries/Heard_Of.shtml.

15 Cezara Dragoi, Thomas A. Henzinger, Helmut Veith, Josef Widder, and Damien Zufferey. A logic-based framework for verifying consensus algorithms. In Kenneth L. McMillan and Xavier Rival, editors, *Verification, Model Checking, and Abstract Interpretation VMCAI 2014*, volume 8318 of *Lecture Notes in Computer Science*, pages 161–181. Springer, 2014. doi:10.1007/978-3-642-54013-4_10.

16 Cezara Dragoi, Thomas A. Henzinger, and Damien Zufferey. Psync: a partially synchronous language for fault-tolerant distributed algorithms. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016*, pages 400–415. ACM, 2016. doi:10.1145/2837614.2837650.

17 Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988. doi:10.1145/42282.42283.

18 Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

19 Felix C. Freiling, Rachid Guerraoui, and Petr Kuznetsov. The failure detector abstraction. *ACM Comput. Surv.*, 43(2):9:1–9:40, 2011. doi:10.1145/1883612.1883616.

20 Eli Gafni. Round-by-round fault detectors: Unifying synchrony and asynchrony (extended abstract). In Brian A. Coan and Yehuda Afek, editors, *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC '98*, pages 143–152. ACM, 1998. doi:10.1145/277697.277724.

21 Rachid Guerraoui and Michel Raynal. The alpha of indulgent consensus. *Comput. J.*, 50(1):53–67, 2007. doi:10.1093/comjnl/bxl046.

22 Michel Hurfin, Achour Mostéfaoui, and Michel Raynal. A versatile family of consensus protocols based on chandra-toueg's unreliable failure detectors. *IEEE Trans. Computers*, 51(4):395–408, 2002. doi:10.1109/12.995450.

23 Igor Konnov, Helmut Veith, and Josef Widder. SMT and POR beat counter abstraction: Parameterized model checking of threshold-based distributed algorithms. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015*, volume 9206 of *Lecture Notes in Computer Science*, pages 85–102. Springer, 2015. doi:10.1007/978-3-319-21690-4_6.

24 Igor V. Konnov, Marijana Lazic, Helmut Veith, and Josef Widder. A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017*, pages 719–734. ACM, 2017. URL: http://dl.acm.org/citation.cfm?id=3009860.

25 Igor V. Konnov, Helmut Veith, and Josef Widder. On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. *Inf. Comput.*, 252:95–109, 2017. doi:10.1016/j.ic.2016.03.006.

26 Jure Kukovec, Igor Konnov, and Josef Widder. Reachability in parameterized systems: All flavors of threshold automata. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory, CONCUR 2018*, volume 118 of *LIPIcs*, pages 19:1–19:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.19.

27 Ognjen Maric. *Formal Verification of Fault-Tolerant Systems*. PhD thesis, ETH Zurich, 2017.

28 Ognjen Maric, Christoph Sprenger, and David A. Basin. Cutoff bounds for consensus algorithms. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017*, volume 10427 of *Lecture Notes in Computer Science*, pages 217–237. Springer, 2017. doi:10.1007/978-3-319-63390-9_12.

29 Yoram Moses. Knowledge in distributed systems. In *Encyclopedia of Algorithms*, pages 1051–1055. Springer, 2016. doi:10.1007/978-1-4939-2864-4_606.

30 Yoram Moses and Sergio Rajsbaum. A layered analysis of consensus. *SIAM J. Comput.*, 31(4):989–1021, 2002. doi:10.1137/S0097539799364006.

31 Achour Mostéfaoui and Michel Raynal. Solving consensus using chandra-toueg's unreliable failure detectors: A general quorum-based approach. In Prasad Jayanti, editor, *Distributed Computing, 13th International Symposium*, volume 1693 of *Lecture Notes in Computer Science*, pages 49–63. Springer, 1999. doi:10.1007/3-540-48169-9_4.

32 Michel Raynal and Julien Stainer. Synchrony weakened by message adversaries vs asynchrony restricted by failure detectors. In Panagiota Fatourou and Gadi Taubenfeld, editors, *ACM Symposium on Principles of Distributed Computing, PODC '13*, pages 166–175. ACM, 2013. doi:10.1145/2484239.2484249.

33 Olivier Rütti, Zarko Milosevic, and André Schiper. Generic construction of consensus algorithms for benign and byzantine faults. In *Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2010*, pages 343–352. IEEE Computer Society, 2010. doi:10.1109/DSN.2010.5544299.

34 Yee Jiun Song, Robbert van Renesse, Fred B. Schneider, and Danny Dolev. The building blocks of consensus. In Shrisha Rao, Mainak Chatterjee, Prasad Jayanti, C. Siva Ram Murthy, and Sanjoy Kumar Saha, editors, *Distributed Computing and Networking, 9th International Conference, ICDCN 2008*, volume 4904 of *Lecture Notes in Computer Science*, pages 54–72. Springer, 2008. doi:10.1007/978-3-540-77444-0_5.

35 Ilina Stoilkovska, Igor Konnov, Josef Widder, and Florian Zuleger. Verifying safety of synchronous fault-tolerant algorithms by bounded model checking. In Tomás Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019*, volume 11428 of *Lecture Notes in Computer Science*, pages 357–374. Springer, 2019. doi:10.1007/978-3-030-17465-1_20.

36 Marcelo Taube, Giuliano Losa, Kenneth L. McMillan, Oded Padon, Mooly Sagiv, Sharon Shoham, James R. Wilcox, and Doug Woos. Modularity for decidability of deductive verification with applications to distributed systems. In Jeffrey S. Foster and Dan Grossman, editors, *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018*, pages 662–677. ACM, 2018. doi:10.1145/3192366.3192414.

37 Tatsuhiro Tsuchiya and André Schiper. Verification of consensus algorithms using satisfiability solving. *Distributed Computing*, 23(5-6):341–358, 2011. doi:10.1007/s00446-010-0123-3.

38 Robbert van Renesse, Nicolas Schiper, and Fred B. Schneider. Vive la différence: Paxos vs. viewstamped replication vs. zab. *IEEE Trans. Dependable Sec. Comput.*, 12(4):472–484, 2015. doi:10.1109/TDSC.2014.2355848.

39 Klaus von Gleissenthall, Nikolaj Bjørner, and Andrey Rybalchenko. Cardinalities and universal quantifiers for verifying parameterized systems. In Chandra Krintz and Emery Berger, editors, *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016*, pages 599–613. ACM, 2016. doi:10.1145/2908080.2908129.

40 Josef Widder and Ulrich Schmid. The theta-model: achieving synchrony without clocks. *Distributed Computing*, 22(1):29–47, 2009. doi:10.1007/s00446-009-0080-x.

41 James R. Wilcox, Doug Woos, Pavel Panchekha, Zachary Tatlock, Xi Wang, Michael D. Ernst, and Thomas E. Anderson. Verdi: a framework for implementing and formally verifying distributed systems. In David Grove and Steve Blackburn, editors, *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 357–368. ACM, 2015. doi:10.1145/2737924.2737958.

42 Doug Woos, James R. Wilcox, Steve Anton, Zachary Tatlock, Michael D. Ernst, and Thomas E. Anderson. Planning for change in a formal verification of the raft consensus protocol. In Jeremy Avigad and Adam Chlipala, editors, *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, pages 154–165. ACM, 2016. doi:10.1145/2854065.2854081.