A Logic-based Framework for Verifying Consensus Algorithms*

Cezara Drăgoi[1], Thomas A. Henzinger[1], Helmut Veith[2], Josef Widder[2], and Damien Zufferey[3]**

[1] IST Austria
[2] TU Wien, Austria
[3] MIT CSAIL

Abstract. Fault-tolerant distributed algorithms play an important role in ensuring the reliability of many software applications. In this paper we consider distributed algorithms whose computations are organized in rounds. To verify the correctness of such algorithms, we reason about (i) properties (such as invariants) of the state, (ii) the transitions controlled by the algorithm, and (iii) the communication graph. We introduce a logic that addresses these points, and contains set comprehensions with cardinality constraints, function symbols to describe the local states of each process, and a limited form of quantifier alternation to express the verification conditions. We show its use in automating the verification of consensus algorithms. In particular, we give a semi-decision procedure for the unsatisfiability problem of the logic and identify a decidable fragment.We successfully applied our framework to verify the correctness of a variety of consensus algorithms tolerant to both benign faults (message loss, process crashes) and value faults (message corruption).

1 Introduction

Fault-tolerant distributed algorithms play a critical role in many applications ranging from embedded systems [12] to data center management [8, 14]. The development of these algorithms has not benefited from the recent progress in automated reasoning and the vast majority of the correctness proofs of these algorithms is still written by hand. A central problem that these algorithms solve is the consensus problem in which distributed agents have initial values and must eventually decide on some value. Moreover, processes must agree on a common value from the set of initial values, even in environments that contain faults and uncertainty in the timing of events. Charron-Bost and Schiper [10] introduced the *heard-of* model as a common framework to model different assumptions on the environment, and to express the most relevant consensus algorithms from the literature. We introduce a new logic $\mathbb{CL}$ tailored for the heard-of model.

* Supported by the National Research Network RiSE of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through grant PROSEED and by the ERC Advanced Grant QUAREM.

** Damien Zufferey was at IST Austria when this work was done.

The heard-of model is a round-based computational model: conceptually, processes operate in lock-step, and distributed algorithms consist of rules that determine the new state of a process depending on the state at the beginning of the round and the messages received by the process in the current round. The work in [10] introduces the notion of *heard-of* set $HO(p, r)$, which contains the processes from which some process p may receive messages in a given round r. Without restricting the heard-of sets, it could be the case that they are all empty, i.e., that there is no communication, and it is obvious that no interesting distributed computing problem can be solved. In [10] a way to describe meaningful communication is introduced, namely via communication predicates that constrain the heard-of sets in the computation. For instance, in a system consisting of n processes, the communication predicate $\forall r \forall p. |HO(p, r)| > n/2$ states that in all rounds all processes can receive messages from a majority of processes. As is the case in this example, the quantification over rounds is typically used in a way that corresponds to a fragment of linear temporal logic, using only simple combinations of the "globally" and "finally" operators. We can thus eliminate the round numbers and rewrite the above example as $\Box(\forall p. |HO(p)| > n/2)$, and call terms like $\forall p. |HO(p)| > n/2$ *topology predicates*, as they restrict the communication graph in a round. It is demonstrated in [10] that many consensus algorithms from the literature can be expressed in this framework. These algorithms are correct only for specific communication predicates.

Our goal is to automate Hoare-style reasoning for distributed algorithms in the heard-of model. To this end, we have to define a logic that has a semi-decision procedure for satisfiability, and is able to capture properties of the states and the effect of the transitions. For instance, our logic must be able to capture topology predicates such as

"each process receives messages from at least $n - t$ processes,"

where n and t are integer variables that model the parameters of the system, such as the number of processes and faulty processes. Moreover, the logic should describe the values of the variables manipulated by the processes. For example

"if a process p decides on a value v, then a majority of processes currently have v stored in their `x` variable."

Finally, we have to capture the transitions of the algorithms, for instance

"all processes that receive a value v from more than two thirds of the processes, set variable `x` to v."

We thus need a logic that allows universal quantification over processes, defining sets of processes depending on the values of their variables, and linear constraints on the cardinalities of such sets of processes. These constraints can be expressed in first order logic, but since the satisfiability problem is undecidable, we need to find a logic that strikes a balance between expressiveness and decidability.

Contributions. We introduce a multi-sorted first-order logic called *Consensus verification logic* $\mathbb{CL}$ whose formulas express topology predicates and constrain the values of the processes' local variables using: (1) set comprehensions, (2) cardinality constraints, (3) uninterpreted functions, and (4) universal quantification. To automate the check of verification conditions we introduce a semi-decision procedure for unsatisfiability. This procedure soundly reduces checking the validity of implications between formulas in $\mathbb{CL}$ to checking the satisfiability of a set of formulas in Presburger arithmetics and a set of quantifier-free formulas with uninterpreted function symbols. The latter two have a decidable satisfiability problem. Furthermore, we have identified a fragment of the logic for which the satisfiability problem is decidable. The proof is based on a small model argument. We have successfully applied the semi-decision procedure to a number of consensus algorithms from the literature. In particular, we have applied it to all algorithms from [10], which surveys the most relevant (partially synchronous) consensus algorithms in the presence of benign faults, including a variant of Paxos. In addition we applied it to the algorithms from [4], which tolerate value faults, and to a basic synchronous consensus algorithm from [19].

2 Fault-tolerant distributed algorithms in the HO-model

In this section, we present the class of distributed algorithms we want to verify. These are algorithms in the *heard-of* model of distributed computations [10]. In the following, we introduce an adaptation of the heard-of model, suitable for automated verification. Distributed algorithms consist of n processes which interact by message passing, where n is a parameter. The executions are organized in rounds, and we model each round to consist of two transitions.

In the first transition, called *environment transition*, processes communicate by exchanging messages and intuitively an adversary, called *environment*, determines for each process the set of processes it receives messages from, i.e., its heard-of set. In a variant of the heard of model [10], the environment also assigns to each process a coordinator process. In the second transition, called *computation transition*, processes change their local state depending of the messages received in the previous phase. These transitions update disjoint sets of variables: the variables updated by the environment, in the first transition of a round, are called *environment variables*, the variables updated by the processes, in the second transition, are called *computation variables*. In the following we describe the variables of the distributed algorithm and the semantics of the two types of transitions.

Variables: The local variables manipulated by the distributed algorithm are of type `process`, `set of processes` or of data types, e.g., `integer` or `boolean`. The variables of type process and sets of processes are the environment variables, denoted *EVars*. The heard-of set of a process is represented by a local variable of type set of processes. Similarly, the coordinator of a process is represented by a local variable of type process. The variables of data types are called computation variables, denoted *CVars*. For some distributed algorithms, we use global

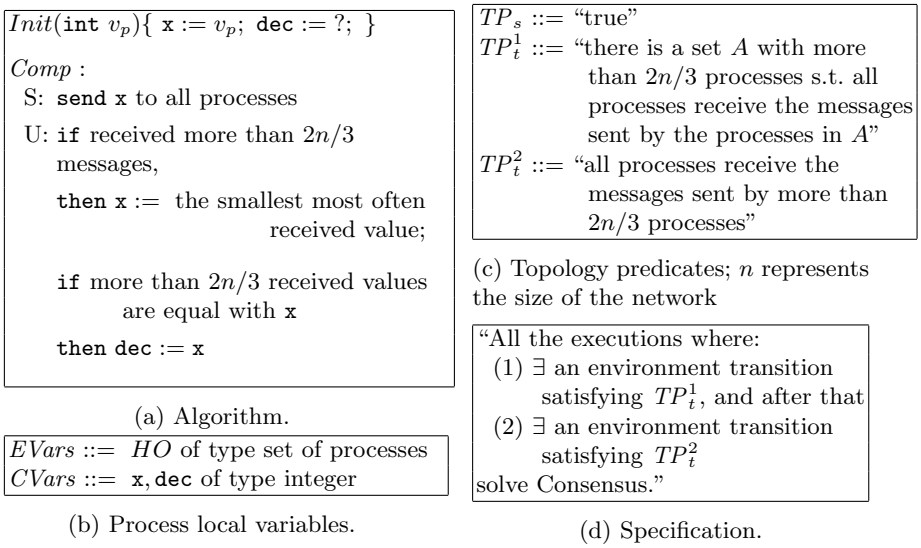


Fig. 1: A round based algorithm in the HO-model that solves Consensus

variables, *GVars*, of integer type to model round numbers. For simplicity of presentation, although of data type, we consider the global variables as environment variables that are deterministically incremented in the environment transitions.

Environment transitions: The environment transitions assigns non-deterministically values to the environment variables of each process.

Computation transition: Computation transitions assign values to the local computation variables of processes. These assignments are guarded by `if-then-else` statements. The latter contain conditions over the local state of the process and the messages received. In our view of the heard-of model we regard messages as values of the local variables of data type of other processes. The set of messages received by a process is determined by the value of its environment variables (*HO*-sets) and the `send` statements executed by the other processes. These statements are of the form "`send` *var* `to` *destination*", e.g. "`send x to all processes`" or "`send x to coordinator`'; they are parametrized by the variables sent, `x`, and the destination processes. More precisely, a process p receives `x` from process q, if q is in the heard-of set of p, and q executes "`send x`" and p is a destination process of this send statement.

Executions: A state of the distributed algorithm is defined by an n-tuple of local process states, and a valuation for the global variables, if there are any. The local state of a process is defined by a valuation of its variables. A computation starts with an initialization round, $Init$, followed by a sequence of rounds, $Comp$. The executions of a typical distributed algorithm are sequences of the form $[p_1.Init(v_1)|| \dots ||p_n.Init(v_n)];\ \big(Env; [p_1.Comp|| \dots ||p_n.Comp];\big)^*$ where Env is an environment transition, $Init$ and $Round$ are defined in Fig. 1, n is the number of processes, $||$ is the parallel composition, $p.R$ states that process p is ex-

ecuting R, '$*$' is the Kleene iteration of the sequential composition, and v_i, for $1 \leq i \leq n$, are integers different from a distinguished integer denoted by '?'.

Example 1. The distributed algorithm in Fig. 1 consists of n processes, each of them having two local variables `x` and `dec` of integer type, and one environment variable, the HO-set. The *computation* transitions are given in Fig. 1a. For each process, the $Init$ transition initializes `dec` to a special value '?' and `x` to an input value. In the other rounds, all processes execute $Comp$. Given a process p, the values of the `x` variables of each process q in $HO(p)$ defines a multiset. It corresponds to the messages received by p.

The first `if` statement means that if p receives messages from more than two thirds of the processes, it updates its local variable `x` to the minimal most often received value. If the condition does not hold, the value of `x` stays unchanged. As the HO-set at different processes may differ, it can be that only some processes update `x`. In the second `if` statement, a process p updates the value of the variable `dec` if it received the same value from more then two thirds of the processes. As two thirds of the processes have the same value, there is a majority around this value.

3 Verification of distributed algorithms

Specifying consensus. Intuitively, a distributed algorithm solves consensus if starting from an initial state where each process p has a value, it reaches a state where all the processes agree on one of the initial values. More precisely, consensus is the conjunction of four properties: *agreement*, no two process decide differently, *validity*, if all processes start with v then v is the only possible decision, *irrevocability*, any decision is irrevocable, and *termination*, eventually all processes decide. It is well-known from literature [22] that consensus cannot be solved if the environment transitions are not restricted. Hence, the specifications we consider are actually conditional. In the literature, the conditions are given in natural language and we express them with *topology predicates* and temporal logic formulas over these predicates. More precisely, topology predicates are conditions on the environment variables. We use topology predicates to restrict the effect of an environment transition, i.e., they restrict the domain of the non-deterministic assignments. To restrict the environment transitions in an execution, we use very simple LTL formulas: we consider conjunctions, where the first conjunct has the form $\Box\phi$, and the second conjunct is of the form $\Diamond(\phi_1 \wedge \Diamond(\phi_2 \wedge \Diamond(\ldots \wedge \Diamond(\phi_\ell))))$, where ϕ, ϕ_1, $\ldots$ ϕ_ℓ are topology predicates.

Example 2. The system in Fig. 1 solves Consensus by making all processes agree on the valuation of `dec`. Its specification is given in Fig. 1d. It uses three topology predicates, TP_s, TP_t^1 and TP_t^2, given in Fig. 1c. In temporal logic parlance, agreement can be stated as $\Box\, Agrm$, where $Agrm$ says that

"for any two processes p, q, either one of them has not decided, i.e., $\texttt{dec} = ?$ or they decide the same value, i.e., $\texttt{dec}(p) = \texttt{dec}(q) \neq ?$" (1)

Termination can be stated as $\Diamond\, Term$, where $Term$ says that "for all processes p, $\mathtt{dec}(p) \neq ?$". To ensure termination, the distributed algorithm in Fig. 1 requires the existence of two specific rounds satisfying the topology predates TP_t^1 and TP_t^2. The specification is then given by $\Box\, TP_s \Rightarrow \Box\, Agrm$ and

$$\Big(\Box\, TP_s \wedge \big(\Diamond\,(TP_t^1 \wedge \Diamond\, TP_t^2)\big)\Big) \Rightarrow \Diamond\, Term.$$

Invariant checking for distributed algorithms. We consider a logic-based framework to verify that a distributed algorithm satisfies its specification, where formulas represent sets of states or binary relations between states.

To prove the safety properties, i.e., agreement, validity, and irrevocability[4], we use the *invariant checking* approach, i.e., given a formula Inv_s that describes a set of states of the system, we check that Inv_s is an *inductive invariant* for the set of computations where all states satisfy the topology predicate TP_s and that Inv_s implies the three safety properties of consensus. The proof that Inv_s is an inductive invariant reduces to checking that the initial states of the system satisfy Inv_s and checking that the following holds:

$$\big(Inv_s(\boldsymbol{p}, \boldsymbol{e}, \boldsymbol{a}) \wedge TP_s(\boldsymbol{p}, \boldsymbol{e}) \wedge TR(\boldsymbol{p}, \boldsymbol{e}, \boldsymbol{e'}, \boldsymbol{a}, \boldsymbol{a'})\big) \Rightarrow Inv_s(\boldsymbol{p}, \boldsymbol{e'}, \boldsymbol{a'})$$

where $\boldsymbol{p}$ is the vector of processes, $\boldsymbol{e}$ is the vector of environment variables, $\boldsymbol{a}$ is the vector of computation variables, $TP_s(\boldsymbol{p}, \boldsymbol{e})$ is a topology predicate, and $TR(\boldsymbol{p}, \boldsymbol{e}, \boldsymbol{e'}, \boldsymbol{a}, \boldsymbol{a'})$ is the transition relation associated with an environment transition or a computation transition (unprimed and primed variables represent the value of the variables before and after a transition, respectively).

In our example, the invariant Inv_s states that

> *"no process has decided or there is a value v such that a majority of processes store the value v in their local variable* `x` *and all processes that have decided chose v as their decision value".* (2)

To prove termination, our technique targets specifications that require a bounded number of constrained environment transitions. W.l.o.g. let r_1 and r_2 be the special rounds required for termination such that r_1 happens before r_2. For simplicity of presentation, we assume that both rounds satisfy the same topology predicate TP_t. To prove termination, the user must provide an inductive invariant, denoted Inv_t, that holds between the two special rounds, that is:

$$\big(Inv_s(\boldsymbol{p}, \boldsymbol{e}, \boldsymbol{a}) \wedge TP_t(\boldsymbol{p}, \boldsymbol{e}) \wedge TP_s(\boldsymbol{p}, \boldsymbol{e}) \wedge TR(\boldsymbol{p}, \boldsymbol{e}, \boldsymbol{e'}, \boldsymbol{a}, \boldsymbol{a'})\big) \Rightarrow Inv_t(\boldsymbol{p}, \boldsymbol{e'}, \boldsymbol{a'})$$

$$\big(Inv_t(\boldsymbol{p}, \boldsymbol{e}, \boldsymbol{a}) \wedge TP_s(\boldsymbol{p}, \boldsymbol{e}) \wedge TR(\boldsymbol{p}, \boldsymbol{e}, \boldsymbol{e'}, \boldsymbol{a}, \boldsymbol{a'})\big) \Rightarrow Inv_t(\boldsymbol{p}, \boldsymbol{e'}, \boldsymbol{a'})$$

Moreover, this invariant has to be strong enough to achieve termination when the second special round happens, that is:

$$\big(Inv_t(\boldsymbol{p}, \boldsymbol{e}, \boldsymbol{a}) \wedge TP_t(\boldsymbol{p}, \boldsymbol{e}) \wedge TP_s(\boldsymbol{p}, \boldsymbol{e}) \wedge TR(\boldsymbol{p}, \boldsymbol{e}, \boldsymbol{e'}, \boldsymbol{a}, \boldsymbol{a'})\big) \Rightarrow Term(\boldsymbol{p}, \boldsymbol{e'}, \boldsymbol{a'}).$$

[4] Irrevocability can be stated as a property of the transition relation. It requires the use of a relational semantics for the round computations.

In our running example, the invariant for termination Inv_t is a stronger version of the safety invariant, and states that "there exists a value v such that the local variable `x` of any process equals v".

4 Consensus verification logic $\mathbb{CL}$

In this section, we introduce our logic $\mathbb{CL}$ that formalizes topology predicates, state properties, and the transition relation. We first introduce a graph-based representation for the states of the distributed algorithms we consider. Then, we define the syntax and semantics of our logic, whose formulas are interpreted over the graph-based representation.

4.1 Graph-based representation of states

We model states by *network graphs*, where each node represents a process. Node and link labels correspond to the values of the computation variables and environment variables, respectively. Formally, network graphs are tuples $G = (N, E, L_N, L_E)$, where N is a finite set of nodes, $L_N : N \times \mathit{CVars} \to \mathcal{D}$ defines a labeling of nodes with values from a potentially unbounded domain $\mathcal{D}$, E is a set of edges, and $L_E : E \rightharpoonup 2^{\mathit{EVars}}$ defines a labeling of the edges. For any environment variable $\texttt{ev} \in \mathit{EVars}$ of `process` type, the edges labeled by `ev` define a *total* function over the nodes in the graph (i.e., each node has exactly one successor defined by an edge labeled by `ev`). The heard-of sets are represented by variables of type `set of processes`; they do not define a total function because a node can have multiple or no successors w.r.t. the label HO.

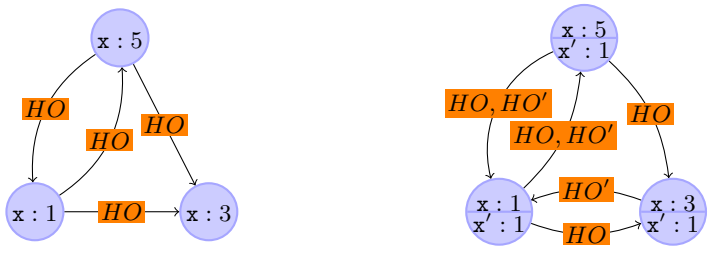
A *state* of a distributed algorithm is a pair $C = (G, \nu)$, where G is a network graph and $\nu : \mathit{GVars} \to \mathcal{D}$ is a valuation of the global variables. Relations between two network states of the same system are represented by pairs (G, ν), where the vocabulary of the labels is doubled by introducing their primed versions. As we are interested in relations between states that belong to the same execution, the two states contain exactly the same set of processes.

Fig. 2a shows a state with three processes of the algorithm in Fig. 1, and Fig. 2b shows a relation between two states of the same algorithm. For simplicity, we draw only the labeled edges and omit the `dec` variable.

4.2 Syntax and semantics

We define a multi-sorted first-order logic, called *Consensus verification logic* $\mathbb{CL}$, to express properties of sets of states (e.g., invariants) or relations between states (transition relations). The syntax of the logic is given in Fig. 3. The logic has four sorts: process, denoted P, sets of processes, denoted 2^P, integers, denoted $\mathbb{Z}$, and data, denoted D. We write $F[\varphi(*)](p)$ instead of $F(p) \cap \{q \mid \varphi(q)\}$.

The models of a formula in $\mathbb{CL}$ are pairs (G, μ), where $G = (N, E, L_N, L_E)$ is a network graph and μ is a valuation of the free variables. In the following, we describe the semantics of $\mathbb{CL}$ formulas and their use. We use the convention that



(a) (b)

Fig. 2: Sample state (a) and relation between states (b)

	Sort P	Sort 2^P	Sort D	Sort $\mathbb{Z}$
Function symbols	$f : P \to P$	$F : P \to 2^P$	$\mathtt{x} : P \to D$	$\lvert\cdot\rvert : 2^P \to \mathbb{Z}$
Variables	$p,\ q$	$S,\ A$	$v,\ \Theta$	$N,\ n$
Terms	$t_P ::= p,\ q,\ f(p)$	$t_S ::= S, F(p),\ t_S \cap t_S$	$t_D ::= v,\ \mathtt{x}(p)$	$t_{\mathbb{Z}} ::= \lvert t_S\rvert,\ N,\ n$
Atomic formulas	$\varphi_P ::= t_P^1 = t_P^2$ $t_P \in t_S,$	$\varphi_S ::= t_S^1 \subseteq t_S^2$	$\varphi_D ::= t_D^1 \leq t_D^2$	$\varphi_{\mathbb{Z}}$ ::= linear constraint over $t_{\mathbb{Z}}$

Set comprehensions $\{q \mid \forall \boldsymbol{t}.\, \varphi(q, \boldsymbol{t}, \boldsymbol{p})\}$, where $\varphi ::= \varphi_P \mid \varphi_D \mid \varphi_{\mathbb{Z}} \mid \neg\varphi \mid \varphi \wedge \varphi$

Universally quantified formulas $\psi^{\forall}(\boldsymbol{S}) :: = \forall \boldsymbol{p}.\, \mathbb{B}(\{p \in S \mid p \in \boldsymbol{p}, S \in \boldsymbol{S}\}) \Rightarrow \mathbb{B}^+(\varphi_D)$, where

$\boldsymbol{p}$ is a set of process variables, $\boldsymbol{S}$ is a set of set variables and given a set of formulas Γ, $\mathbb{B}(\Gamma)$, resp. $\mathbb{B}^+(\Gamma)$, is a boolean combination, resp., positive boolean combination, of formulas in Γ

$\psi ::= \varphi_P \mid \varphi_S \mid \varphi_{\mathbb{Z}} \mid \varphi_D \mid \psi^{\forall} \mid \psi \wedge \psi \mid \neg\psi,$ $\psi_{\mathbb{CL}} ::= \psi \mid \exists p.\, \psi_{\mathbb{CL}} \mid \exists v.\, \psi_{\mathbb{CL}}$

where set comprehensions can be used as set terms

Fig. 3: Syntax of $\mathbb{CL}$ formulas, defined by $\psi_{\mathbb{CL}}$.

the global variables correspond to free variables of formulas. The satisfaction relation is denoted by $G \models_\mu \varphi$. The interpretation of a term t w.r.t. (G, μ) is denoted by $[\![t]\!]_{(G,\mu)}$.

Atomic formulas over terms of sort process: The terms of sort P are built using a set of function symbols Σ_{pr} of type $P \to P$. They are interpreted as nodes in the graph, e.g., for any variable $p \in P$, $[\![p]\!]_{(G,\mu)}$ is a node in the graph G. The interpretation of the function symbols is defined by the labeled edges, i.e., $[\![f(p)]\!]_{(G,\mu)} = u$ iff the graph G contains an edge $([\![p]\!]_{(G,\mu)}, u)$ labeled by f. The only predicate over terms of type P is equality.

We use the function symbols in Σ_{pr} for two purposes. First, they represent the values of local environment variables of type process, such as the coordinator

of a process. Second, we use them to model processes in the heard-of sets with distinguished local states, such as the processes storing the minimal value, or the value with the most occurrences in the considered set.

Atomic formulas over terms of sort data: The terms of sort D are interpreted as values of the data domain $\mathcal{D}$. The node labels in G, i.e., the values of the computation variables, are represented in the logic by a set of function symbols $\mathtt{x} : P \to D$, one for each node label/computation variable. That is, $[\![\mathtt{x}(p)]\!]_{(G,\mu)} = d$ iff $d \in \mathcal{D}$ is the label $\mathtt{x}$ of the node $[\![p]\!]_{(G,\mu)}$, i.e., $L_N([\![p]\!]_{(G,\mu)}, \mathtt{x}) = d$. We assume that the domain $\mathcal{D}$ is totally ordered. The predicates over data terms are non-strict comparison and equality.

Atomic formulas over terms of sort set: The terms of sort 2^P are interpreted as sets of processes, i.e., sets of nodes in the graph. They are built using a set of function symbols Σ_{set} of type $P \to 2^P$. For any function symbol $F : P \to 2^P$ in Σ_{set}, $[\![F(p)]\!]_{(G,\mu)}$ is a set of nodes from N such that $u \in [\![F(p)]\!]_{(G,\mu)}$ iff F is one of the labels of the edge $(u, [\![p]\!]_{(G,\mu)}) \in E$. The heard-of sets are modeled using a function symbol $HO \in \Sigma_{set}$, where $[\![HO(p)]\!]_{(G,\mu)}$ is the set of nodes representing the processes $[\![p]\!]_{(G,\mu)}$ hears from. The logic contains the inclusion predicate over set terms and the membership predicate over process and set terms.

Atomic formulas over terms of sort integer: The atomic formulas over $\mathbb{Z}$-terms are linear inequalities and they constrain the cardinality of the set terms. We consider a distinguished integer variable n, which is interpreted as the number of processes in the network. For example, $|HO(p)| > 2n/3$ states that the process p receives messages sent by more than two thirds of the processes, and $\big|HO\big[\mathtt{x}(*) = \mathtt{x}(p)\big](p)\big| > 2n/3$ states that the value $\mathtt{x}(p)$ is received more than $2n/3$ times by process p.

One of the key features of the logic are the *set comprehensions*. They are used in the invariants to state that a majority is formed around one value, in the topology predicates to identify the set of processes that every one hears from, and in the transition relation to identify the processes that will update their local state. A set comprehension is defined by $\{q \mid \rho(q)\}$, where ρ is a (universally quantified) formula that contains at least one occurrence of the variable q (representing processes in the set). For ease of notation, we associate with each set comprehension a unique set variable used in a formula as a macro for its definition. The interpretation of a set comprehension is $[\![\{q \mid \rho(q)\}]\!]_{(G,\mu)} = \{u \in N \mid G \models_{\mu[q \leftarrow u]} \rho(q)\}$.

Set comprehensions with quantifier-free formulas: Typically, invariants identify sets of processes whose local variables have the same value. For example, the invariant Inv_s in (2) is defined using the set of processes whose local variable $\mathtt{x}$ equals v, i.e., $S_V = \{q \mid \mathtt{x}(q) = v\}$. Topology predicates are also expressed using set comprehension: the two topology predicates from Fig. 1c are expressed in $\mathbb{CL}$ by:

$$TP_t^1 ::= |A| > 2n/3 \ \wedge \ |S_A| = n, \text{ with } S_A = \{q \mid HO(q) = A\}$$
$$TP_t^2 ::= |S_{HO}| = n, \text{ with } S_{HO} = \{q \mid |HO(q)| > 2n/3\},$$

where A is a set variable, S_A is the set of processes whose heard-of set equals A, and S_{HO} is the set of processes that receive from more than $2n/3$ processes.

Set comprehensions with universally quantified formulas: Typical examples of such set comprehensions used in topology predicates are: the kernel $K = \{q \mid \forall t.\, q \in HO(t)\}$, i.e., the set of processes every one hears from and $S_{no\_split} = \{q \mid \forall t. |HO(t) \cap HO(q)| \geq 1\}$, which is the set of processes that share some received message with any other process in the network.

Set comprehensions are also used to select the processes that update their local state. Typically, the value assigned to some local variable is chosen from the received ones, e.g., the minimal received value, or the minimal most often received value. To express such updates, the process in the HO-set that holds such a value is represented as the value of a function symbol in Σ_{pr}. For example, the first update from the algorithm in Fig. 1a can be written as $\mathtt{x}'(p) = \mathtt{x}(mMoR(p))$, where $mMoR(p)$ is interpreted as a process q s.t. $\mathtt{x}(q)$ is the minimal most often received value by p. This constraint over the interpretation of $mMoR$ can be expressed by $|S| = n$ (we assume that all processes have sent the value of their $\mathtt{x}$ variable), where

$$S = \left\{ \begin{array}{l} q \mid \forall t.\, t \in HO(q) \Rightarrow \\ \quad \left(\begin{array}{r} |HO[\mathtt{x}(*) = \mathtt{x}(mMoR(q))](q)| = |HO[\mathtt{x}(*) = \mathtt{x}(t)](q)| \Rightarrow \\ \mathtt{x}(mMoR(q)) \leq \mathtt{x}(t) \\ \wedge |HO[\mathtt{x}(*) = \mathtt{x}(mMoR(q))](q)| \geq |HO[\mathtt{x}(*) = \mathtt{x}(t)](q)| \end{array} \right) \end{array} \right\} \tag{3}$$

Above, S represents the set of processes q s.t. $\mathtt{x}(mMoR(q))$ is interpreted as the minimal most often received value by q. If $|S| = n$, i.e., S contains all the processes in the network, then for all processes q, the $\mathtt{x}$ variable of $mMoR(q)$ equals the minimal most often received value by q.

Universally quantified formulas: The universally quantified formulas in $\mathbb{CL}$ are implications, where (1) quantification is applied only over process variables, (2) the left hand side of the implication is a boolean combination of membership constraints, and (3) the right hand side of the implication is a positive boolean combination (without negation) of atomic formulas over data. For example, the transition relation for the algorithm in Fig. 1 is expressed by

$$\begin{array}{r} TR \;= \forall p.\, p \in S_{HO} \Rightarrow \mathtt{x}'(p) = mMoR(p) \;\wedge\; \forall p.\, p \in S_{HV} \Rightarrow \mathtt{dec}'(p) = \mathtt{x}'(p) \;\wedge \\ \forall p.\, p \notin S_{HO} \Rightarrow \mathtt{x}'(p) = \mathtt{x}(p) \;\wedge\; \forall p.\, p \notin S_{HV} \Rightarrow \mathtt{dec}'(p) = \mathtt{dec}(p), \end{array} \tag{4}$$

where $S_{HV} = \{q \mid |HO[\mathtt{x}(*) = \mathtt{x}'(q)](q)| > 2n/3\}$ and S_{HO} is defined above.

The state properties in the definition of consensus, e.g., $Agrm$ given by (1) in Sec. 3, are expressed using universally quantified formulas:

$$Agrm \;=\; |S| = n \wedge \forall p, q.\, p, q \in S \Rightarrow \mathtt{dec}(p) = \mathtt{dec}(q). \tag{5}$$

Remark 1. The formulas that express the guarded assignments, the inductive invariants, and the properties that define consensus, are in the form of universally-quantified implications. The left-hand side of these implications is typically more involved. To express these formulas, $\mathbb{CL}$ restricts the syntax of universally-quantified implications in a way that is sufficient to express the formulas we

encountered. Note that these constraints on the use of universal variables can be overpassed using set comprehensions, e.g., $\forall t, q.\, \mathtt{x}(t) \neq \mathtt{x}(q)$ is equivalent to $S = \{q \mid \forall t.\, \mathtt{x}(t) \neq \mathtt{x}(q)\} \wedge |S| = n$.

Finally, $\mathbb{CL}$ formulas are existentially-quantified boolean combinations of atomic formulas and universally quantified formulas.

To conclude let us formalize the definition of the invariants, Inv_s and Inv_t given in Sec. 3, required to prove the correctness of the system in Fig. 1.

$$\begin{array}{rl} Inv_s & = Inv_s^1 \vee \exists v.\, Inv_s^2(v), \text{where} \\ & \qquad Inv_s^1 = \forall q.\, \mathtt{dec}[q] = ? \text{ and} \\ & \qquad Inv_s^2(v) = |S_V| > 2n/3 \wedge \forall q.\, \mathtt{dec}(q) = ? \vee \mathtt{dec}(q) = v = \mathtt{x}(q) \\ Inv_t & = \exists v\, \forall q.\, \mathtt{x}(p) = v \wedge \big(\mathtt{dec}(q) = ? \vee \mathtt{dec}(q) = v = \mathtt{x}(q)\big) \end{array} \tag{6}$$

Verification condition for distributed algorithms are implications between $\mathbb{CL}$ formulas, such as the ones in Sect. 3, where the invariants, transition relations, and properties are expressed in $\mathbb{CL}$.

5 A semi-decision procedure for implications

Classically, checking the validity of a formula is reduced to checking the unsatisfiability of its negation. Since $\mathbb{CL}$ is not closed under negation, the negation of an implication between $\mathbb{CL}$ formulas is not necessarily in $\mathbb{CL}$. In this section, we will present (1) a sound reduction from the validity of an entailment between two formulas in $\mathbb{CL}$ to the unsatisfiability of a formula in $\mathbb{CL}$, (2) a semi-decision procedure for the unsatisfiability problem in $\mathbb{CL}$, and (3) identify a fragment $\mathbb{CL}_{dec}$ of $\mathbb{CL}$ which is decidable.

5.1 Reducing entailment checking in $\mathbb{CL}$ to unsatisfiability

Let us consider the following entailment $\varphi \Rightarrow \psi$ between $\mathbb{CL}$ formulas φ and ψ. There are two reasons why $\varphi \wedge \neg\psi$ might not belong to $\mathbb{CL}$. First, if ψ has a sub-formula of the form $\exists^*\forall^*$, then by negation, the quantifier alternation becomes $\forall^*\exists^*$, which is not allowed in $\mathbb{CL}$. Second, the restricted use of universally quantified variables in $\mathbb{CL}$ is not preserved by negating the constraints on the existential variables of ψ, e.g., if $\psi = \exists p_1, p_2.\, p_1 = p_2$, then its negation is not in $\mathbb{CL}$ because difference constraints between universally quantified variables are not allowed. We define a procedure, which receives as input an implication $\varphi \Rightarrow \psi$ between two formulas in $\mathbb{CL}$. The algorithm we define hereafter builds a new formula ϕ from $\varphi \wedge \neg\psi$. It restricts the interpretation of the universally quantified variables that do not satisfy the syntactical requirements of $\mathbb{CL}$ to terms built over the existentially quantified variables.

Reduction procedure: The formula ϕ is built in three steps. In the first step, the formula $\varphi \Rightarrow \psi$ is transformed into an equivalent formula $\phi_1 = \exists \xi.\, (\varphi_1 \wedge \psi_1)$

where all the existential quantifiers appear at the beginning (φ_1 and ψ_1 are equivalent to φ and $\neg\psi$, respectively, modulo some renaming of existentially-quantified variables; also, ψ_1 is transformed such that no universal quantifier appears under the scope of a negation). The second step consists of identifying the set of universally quantified variables $\boldsymbol{\beta}$ in ψ_1 that appear in sub-formulas not obeying the syntactic restrictions of $\mathbb{CL}$.

In the last step, let $\mathcal{T}(\xi)$ be the set of terms over the variables in ξ that contain at most k occurrences of the function symbols from $\mathbb{CL}$, where k is the maximum number of function symbols from a term of the formula $\varphi \wedge \neg\psi$. Then, ϕ is obtained by restricting the interpretation of the universally quantified variables in $\boldsymbol{\beta}$ to the domain defined by the interpretation of the terms in $\mathcal{T}(\xi)$:

$$\phi = \exists \xi. \bigwedge_{\gamma \in [\boldsymbol{\beta} \rightarrow \mathcal{T}(\xi)]} \left(\varphi_1 \wedge \psi_1 \left[\begin{array}{c} \beta \leftarrow \gamma(\beta) \\ \text{for every } \beta \in \boldsymbol{\beta} \end{array} \right] \right) \tag{7}$$

Lemma 1. *Let $\varphi \Rightarrow \psi$ be an implication between two formulas in $\mathbb{CL}$, and ϕ the formula in (7). The unsatisfiability of ϕ implies the validity of $\varphi \Rightarrow \psi$.*

Note that if ψ has no existentially quantified variables, then the unsatisfiability of ϕ is equivalent to the validity of $\varphi \Rightarrow \psi$.

Rationale: All the state properties used to define consensus, e.g., *Agrm* in (5) from Sec. 4, are expressible using universally-quantified formulas in $\mathbb{CL}$. Thus, for checking that an invariant implies these properties, the reduction procedure is sound and complete. This is not necessarily true for the verification conditions needed to prove the inductiveness of an invariant, that is, verification conditions of the form $\varphi \Rightarrow \psi$, where ψ is an inductive invariant. In the following, we give evidences for the precision of the reduction procedure in these cases.

In systems that solve consensus, all the computations contain a transition after which only one decision value is possible [11]. Therefore, the set of reachable states can be partitioned into two: the states where any value held by a process may become the decision, and the states where there is a unique value v that can be decided; often this corresponds to a majority formed around v. This implies that the inductive invariants are usually a disjunction of two formulas, one for each set of states described above. In the negation of the invariant, the universally quantified variables that do not obey to the restrictions in $\mathbb{CL}$ are those used to express that there is no value on which all processes agree.

In all our examples, the two sets of states are demarcated by the existence of at least one process that has decided.[5] Given invariants in this disjunctive form, to prove them inductive w.r.t. a transition TR, two situations have to be considered: (1) no process has decided before applying TR and at least one process has decided after TR, and (2) some processes decided before applying TR. In (1), to prove the unsatisfiability of $\varphi \wedge \neg\psi$ it is sufficient to map the universally quantified variables in $\neg\psi$ on terms denoting the value of one of the processes

[5] The only exception is the *LastVoting* algorithm, where the demarcation includes also the existence of a process having a local variable (not the decision one) set to true.

that have decided, and in (2), it is sufficient to map them on the terms denoting the values around which a majority was formed before applying TR.

Example 3. To prove that Inv_s, given in (6), is an invariant w.r.t. the transition relation TR given in (4), more precisely the case where no process decided before applying TR, one needs to prove the validity of $(Inv_s^1 \wedge TR) \Rightarrow Inv_s[\mathtt{dec} \leftarrow \mathtt{dec}']$, where $Inv_s[\mathtt{dec} \leftarrow \mathtt{dec}']$ is the obtained from Inv_s by substituting the function symbol $\mathtt{dec}$ with the function symbol $\mathtt{dec}'$. This is equivalent with proving the unsatisfiability of the following formula, where we have expanded the definition of $Inv_s[\mathtt{dec} \leftarrow \mathtt{dec}']$:

$$Inv_s^1 \ \wedge \ TR \ \wedge \ \exists p.\, \mathtt{dec}'(p) \neq ? \ \wedge \ \forall v.\, \underbrace{\neg Inv_s^2(v)[\mathtt{dec} \leftarrow \mathtt{dec}']}_{\rho(v)}. \tag{8}$$

Notice that $\forall v. \neg Inv_s^2(v)[\mathtt{dec} \leftarrow \mathtt{dec}']$ does not belong to $\mathbb{CL}$ because it contains a $\forall\exists$ quantifier alternation. In this case, we soundly reduce the unsatisfiability of (8) to the unsatisfiability of

$$Inv_s^1 \ \wedge \ TR \ \wedge \ \exists p.\, \mathtt{dec}'(p) \neq ? \wedge \rho(v)[v \leftarrow \mathtt{dec}'(p)]. \tag{9}$$

by restricting the interpretation of universally quantified variable v to the value decided by process p, i.e., $\mathtt{dec}'(p)$.

If some processes decided, the term denoting the value around which a majority was formed before applying TR is the existentially quantified v in Inv_s.

5.2 Semi-decision procedure for unsatisfiability

In this section, we present the semi-decision procedure for the unsatisfiability problem in $\mathbb{CL}$. This procedure soundly reduces the unsatisfiability of a $\mathbb{CL}$ formula to the unsatisfiability of a quantifier-free Presburger formula (cardinality constraints) or the unsatisfiability of a formula with uninterpreted functions and order constraints (constraints on data). The satisfiability of these formulas is decidable and checkable using an SMT solver.

We give an overview on the main steps, Step 1 to Step 5, of the semi-decision procedure on an example, before we formalize them.

Overview: Let us consider the formula in (9), stating that no process decided before applying the transition relation TR given in (4), and afterwards two processes decide on different values:

$$\varphi \ = \ \forall t.\, \mathtt{dec}(t) = ? \ \wedge \ TR \ \wedge \ \mathtt{dec}'(p) \neq ? \wedge \mathtt{dec}'(q) \neq ? \wedge \mathtt{dec}'(p) \neq \mathtt{dec}'(q)$$

The semi-decision procedure starts by instantiating universal quantifiers and set comprehension over the free variables of φ. This strengthens the data and cardinality constraints over terms with free variables (see Step 3).

In our example, the cardinality constraints are strengthened by instantiating the universal quantification in TR and the definition of the set comprehension S_{HV}, over the free variables p and q. The processes denoted by p and q

decide in the round described by TR, therefore these variables belong to the set S_{HV}; from the definition of S_{HV}, the value decided by each of them, i.e., $\mathtt{x}'(p)$, resp., $\mathtt{x}'(q)$, was received from at least two thirds of the processes in the network, i.e., $|HO[\mathtt{x}(*) = \mathtt{x}'(p)](p)| > 2n/3$ and $|HO[\mathtt{x}(*) = \mathtt{x}'(q)](q)| > 2n/3$. The semi-decision procedure builds a Presburger formula from the cardinality constrains that use set terms over p and q; the definitions of the sets are abstracted. The obtained formula is $kp > 2n/3 \wedge kq > 2n/3$, where $kp = |HO[\mathtt{x}(*) = \mathtt{x}'(p)](p)|$ and $kq = |HO[\mathtt{x}(*) = \mathtt{x}'(q)](q)|$ (see Step 4). This formula is a satisfiable.

Then, the semi-decision procedure checks the satisfiability of the quantifier-free formula with uninterpreted function symbols defined by the data constraints over terms with free variables (Step 5). In our example this formula is $\mathtt{dec}'(p) = \mathtt{x}'(p) \ \wedge \ \mathtt{dec}'(q) = \mathtt{x}'(q) \ \wedge \ \mathtt{dec}'(p) \neq \mathtt{dec}'(q)$, and is also satisfiable.

Therefore, for $\mathbb{CL}$ formulas, restricting the interpretation of universal quantifiers to free variables is not sufficient to derive contradictions. The reason is that cardinality constraints induce relations between set comprehentions, which are neither captured by the Presburger formula nor the quantifier-free data formula. Notice that due to cardinality constraints, which state that each of the sets $HO[\mathtt{x}(*) = \mathtt{x}'(p)](p)$, resp. $HO[\mathtt{x}(*) = \mathtt{x}'(q)](q)$, contains more then two thirds of the processes in the network, their intersection is non-empty. Therefore there exists a process, r, which belongs to both sets. Instantiating the definitions of these sets over r reveals that $\mathtt{x}(r) = \mathtt{x}'(p)$ and $\mathtt{x}(r) = \mathtt{x}'(q)$, which contradicts the hypothesis that p and q decided on different values.

Thus, if the Presburger formula is satisfiable, it is used to discover relations between set comprehension. This formula is used to check which intersections or differences of set variables are non-empty; for each non-empty region, φ is extended with a free variable representing a process of this region (see Step 4). The semi-decision procedure is restarted using the new formula constructed from φ.

Semi-decision procedure: Let $\varphi = \forall \boldsymbol{y}.\, \psi$ be a $\mathbb{CL}$ formula in prenex normal form, where $\boldsymbol{y}$ is a tuple of process variables. W.l.o.g., we assume that the formula does not contain existential quantifiers, only free variables. Formally, the procedure to check the unsatisfiability of φ iterates over the following sequence of steps:

Step 1: introduce fresh process variables for the application of function symbols over free variables. Let $\varphi_1 = \forall \boldsymbol{y}.\, \psi_1$, be the formula obtained after this step.

Step 2: enumerate truth valuations for set membership over free variables and instantiate set comprehensions. Let $\varphi_2 = \bigvee \forall \boldsymbol{y_2} \psi_2$, where each disjoint corresponds to a truth valuation. Notice that new quantified formulas are introduced in this step. Let $S = \{q \mid \rho(q)\}$ be a set comprehension, where ρ is universally quantified and p a free variable of φ_1. Then, $p \in S$ introduces a new universally quantified formula, i.e., $\rho(p)$, while $p \notin S$ introduces a new existentially quantified formula, i.e., $\neg\rho(p)$. W.l.o.g the existential quantified variables introduces at this step are transformed into free variables, modulo a renaming.

Step 3: instantiate universal quantifiers over the free variables of φ_1. Let $\boldsymbol{p}$ denote the set of free process variables of φ_1 (note that the free variables introduced

in Step 2 are not in $\boldsymbol{p}$). Each disjunct $\forall \boldsymbol{y_2}.\, \psi_2$ of φ_2 is equivalently rewritten as $\psi_{2,\exists} \wedge \forall \boldsymbol{y_2}.\, \psi_2$, where $\psi_{2,\exists} = \bigwedge_{\gamma_e \in [\boldsymbol{y_2} \to \boldsymbol{p}]} \psi_2\,[\gamma_e]$ and $\psi_2\,[\gamma_e]$ is obtained from ψ_2 by substituting each $y \in \boldsymbol{y_2}$ with $\gamma_e(y)$. Let φ_3 denote the obtained formula.

Step 4: enumerate truth valuations for set and cardinality constraints over free variables in φ_3. Let $\mathcal{A}_s(\varphi_3)$ denote the set of atoms that contain the inclusion or the cardinality operator; each disjunct $\psi_{2,\exists} \wedge \forall \boldsymbol{y_2}.\, \psi_2$ of φ_3 is transformed into the equivalent formula

$$\bigvee_{\gamma_s \in [\mathcal{A}_s(\varphi_3) \to \{0,1\}]} \Big(\bigwedge_{\gamma_s(a)=1} a \wedge \bigwedge_{\gamma_s(a)=0} \neg a \wedge \psi_{2,\exists}[\gamma_s] \wedge \forall \boldsymbol{y_2}.\, \psi_2 \Big), \tag{10}$$

where $\psi_{2,\exists}[\gamma_s]$ is obtained from $\psi_{2,\exists}$ by substituting every $a \in \mathcal{A}_s(\varphi_3)$ with $\gamma_s(a)$.

For each disjunct of the formula in (10), let $\mathcal{C}[\gamma_s]$ be a quantifier-free Presburger formula defined as follows:

- let $\mathcal{T}_s(\varphi_3)$ be the set of terms $S_1 \cap S_2$ or $S_1 \setminus S_2$, where S_1 and S_2 are set variables or applications of function symbols of type $P \to 2^P$ in φ_3 (i.e., they do not contain $\cap$);
- let K_s be a set of integer variables, one variable $k[t]$ for each term $t \in \mathcal{T}_s(\varphi_3)$. Each variable $k[t]$ represents the cardinality of the set denoted by t;
- transform each literal a or $\neg a$ with $a \in \mathcal{A}_s(\varphi_3)$ into a linear constraint over the integer variables K_s:
  - if a is a cardinality constraint, then replace every term $|S|$ by the sum of all variables $k[t]$ with $t \in \mathcal{T}_s(\varphi_3)$ of the form $S \cap S'$ or $S \setminus S'$;
  - transform set inclusions into cardinality constraints: for every atom a of the form $S_1 \subseteq S_2$, if $\gamma_s(a) = 1$ (resp., $\gamma_s(a) = 0$), a is rewritten as $k[S_2 \setminus S_1] = 0$ (resp., $k[S_1 \setminus S_2] \geq 1$). The extension to more general atoms that use the inclusion operator is straightforward;
- for any atom $p \in S$ from (10) (chosen in Step 2), add $|S| \geq 1$ to $\mathcal{C}[\gamma_s]$; similar constraints can be added for more general constraints of the form $p \in t_S$.

If all Presburger formulas associated with the disjuncts of (10) are unsatisfiable, then φ is unsatisfiable. Otherwise, the formula in (10) is transformed into an equivalent formula of the form

$$\bigvee_{\substack{\gamma_s \in [\mathcal{A}_s(\varphi_3) \to \{0,1\}],\\ \mathcal{C}[\gamma_s] \text{ satisfiable}}} \Big(\psi_{2,\exists}[\gamma_s] \wedge \big(\bigwedge_{\mathcal{C}[\gamma_s] \Rightarrow k[t] > 0} \exists p_t.\, p_t \in t \big) \wedge \forall \boldsymbol{y_2}.\, \psi_2 \Big) \tag{11}$$

Step 5: Note that all $\psi_{2,\exists}[\gamma_s]$ are quantifier-free formulas with uninterpreted functions and order constraints, for which the satisfiability problem is decidable. If all $\psi_{2,\exists}[\gamma_s]$ are unsatisfiable then φ is unsatisfiable, otherwise the procedure returns to Step 1 by letting φ be the disjunction of all formulas in (11) that have a satisfiable $\psi_{2,\exists}[\gamma_s]$ sub-formula.

5.3 Completeness

We identify a fragment of $\mathbb{CL}$, denoted $\mathbb{CL}_{dec}$, whose satisfiability problem is decidable. The proof is based on a small model argument. The syntactical restrictions in $\mathbb{CL}_{dec}$ are: (1) function symbols of type $P \to P$ are used only in

constraints on data, i.e., they occur only in data terms of the form $\mathtt{x}(f(p))$, (2) there exists no atomic formula that contains two different function symbols of type $P \to 2^P$, (3) cardinality constraints are restricted to atomic formulas of the form $\varphi_{\mathbb{Z}} ::= c * |t_s| \geq exp \;\mid\; |t_s^1| \geq |t_s^2| \;\mid\; exp = c * n + b$, with $b, c \in \mathbb{Z}$, and (4) set comprehensions are defined using conjunctions of quantifier-free atomic formulas or universally quantified formulas of the form:

$$S = \{q \mid \forall \boldsymbol{t}.\, \varphi_P(q, \boldsymbol{t})\} \mid S = \{q \mid \forall \boldsymbol{t}.\, \varphi_{\mathbb{Z}}(q, \boldsymbol{t})\} \mid S = \{q \mid \forall \boldsymbol{t}.\, \boldsymbol{t} \in F(q) \Rightarrow \varphi^{loc}(q, \boldsymbol{t})\}$$
$$\text{with } \varphi^{loc}(q, \boldsymbol{t}) ::= \varphi_D(q, \boldsymbol{t}) \mid \varphi_{\mathbb{Z}}(q, \boldsymbol{t}) \mid \varphi^{loc}(q, \boldsymbol{t}) \wedge \varphi^{loc}(q, \boldsymbol{t}) \mid \neg\varphi^{loc}(q, \boldsymbol{t})$$

where the $\boldsymbol{t}$, q appear only in terms of the form $F[\Pi](q)$, $\mathtt{x}(q)$, $\mathtt{x}(f(q))$, $\mathtt{x}(t)$ with Π an atomic data formula over $\boldsymbol{t}$ of q.

Let S be a set comprehension whose definition uses universal quantification and terms of the form $F(t)$, with t universally quantified. The set S defines a relation between its elements and a potentially unbounded number of processes in the network. They are called *relational set comprehensions*. An example is the kernel of a network, i.e., $K = \{q \mid \forall t.\, q \in HO(t)\}$. In $\mathbb{CL}_{dec}$, the only constraints allowed on relational set comprehensions are lower bounds on their cardinality, i.e., given such a set S, it occurs only in atomic formulas $|S| \geq exp$ or $|S| \geq |t_S|$ under an even number of negations.

The uncoordinated algorithms in [10] are captured by $\mathbb{CL}_{dec}$, including our running example.

Theorem 1. *The satisfiability problem for formulas in $\mathbb{CL}_{dec}$ is decidable.*

Proof (Sketch): The small model property for this fragment of $\mathbb{CL}$ can be stated as follows: for any formula φ, if φ is satisfiable then φ has a model whose size is bounded by a minimal solution of a quantifier-free Presburger formula constructed from φ; the order relation on solutions, i.e., on tuples of integers, is defined component-wise. Note that, in general, there are exponentially-many minimal solutions for a quantifier-free Presburger formula.

The Presburger formula is constructed from φ by applying a modified version of Step 4 from the semi-decision procedure in Sec. 5.2. One starts by considering the Venn diagram induced by the set/process variables used in the formula (process variables are considered singleton sets) and enumerating all possibilities of a region to be empty or not. For each non-empty region and each function symbol of type $P \to P$ (resp., $P \to 2^P$), we introduce a fresh process variable (resp., set variable) representing the value of this function for all the elements in this region. This is possible because it can be proven that if φ has a model of size n then it has also a model of the same size, where all the nodes in the same Venn region share the same value for their function symbols. Then, one enumerates all truth valuations for the cardinality constraints and constructs a Presburger formula encoding these constraints over a larger (exponential) set of integer variables, one for each region of the Venn diagram (this diagram includes also the set/process variables introduced to denote values of function symbols).

Given a bound on the small models, one can enumerate all network graphs of size smaller than this bound, and compute a quantifier-free formula with uninter-

preted functions and order constraints for each one of them. The original formula is satisfiable iff there exists such a quantifier-free formula which is satisfiable.

5.4 Discussion

The semi-decision procedure introduced in Sec. 5.2 is targeting the specific class of verification conditions for consensus. Intuitively, when designing consensus algorithms one wants to avoid that two disjunct sets of processes decide independently of each other, as this may lead to a violation of agreement. There are two ways to avoid it. First, the algorithm can use a topology predicate to enforce that any two HO-sets intersect (no-split). Second, the algorithm can ensure that a decision is made only if "supported" by a *majority* of processes. When we apply the semi-decision procedure on formulas expressing the negation of these two statements, typically it proves them unsatisfiable. It derives a contradiction starting from the assumption that two sets are disjoint due to their definition (by comprehension). In the first case, the contradiction is obtained by exploiting an explicit cardinality constraint on the intersection, i.e., that the cardinality of the intersection is greater than or equal to 1. In the second case, the contradiction is derived from the fact that each of the two sets have cardinality greater than $n/2$ (majority). For this, one needs to enumerate all pairs of sets and check that their cardinality constraints imply non-empty intersection.

For arbitrary formulas in $\mathbb{CL}$ our semi-decision procedure may fail to derive a contradiction, because one may need to explore the exponentially many regions of the Venn diagram that are induced by the sets represented in the formula. For the decidable fragment $\mathbb{CL}_{dec}$, this is done by the decision procedure in Sec. 5.3.

To conclude, our semi-decision procedure targets the specific class of verification conditions needed for consensus. The semi-decision procedure proves the unsatisfiability of formulas that are not in $\mathbb{CL}_{dec}$. Compared to the decision procedure for $\mathbb{CL}_{dec}$, the semi-decision procedure avoids deciding quantifier-free Presburger formulas over an exponential number of variables and computing all minimal solutions of such formulas (which are exponentially many).

6 Evaluation

We have evaluated our framework on several fault-tolerant consensus algorithms taken from [10], [4], and [19]. All algorithms in [10] and [4] fit into our framework. We tested our semi-decision procedure by manually encoding the algorithms, invariants, and properties in the SMT-LIB 2 format and used the Z3 [21] SMT-solver to check satisfiability of the formulas produced by the semi-decision procedure. In the reduction, we inline the minimization problem along the rest of the formula and let Z3 instantiate the universal quantifiers. The results are summarized in Table 1. The files containing the verification conditions are available at `http://pub.ist.ac.at/~zufferey/consensus/`. We give a short description of each algorithm and how it is proven correct in our framework. The consensus algorithms we considered are presented in a way such that several consecutive

Algorithm	coord. (1)	rounds (2)	invariants (3)	VCs (4)	solving (5)
Uniform Voting [10]	×	2	2+2	13	< 0.1s
Coordinated Uniform Voting [10]	✓	3	2+2	10	< 0.1s
Simplified Coordinated Uniform Voting [10]	✓	2	2+2	8	< 0.1s
One Third Rule [10]	×	1	1+1	8	< 0.1s
Last Voting [10]	✓	4	1+3	15	1s
$\mathcal{A}_{T,E}$ [4]	×	1	1+2	10	< 0.1s
$\mathcal{U}_{T,E}$ [4]	×	2	2+2	9	< 0.1s
FloodMin [19, Chapter 6.2.1]	×	1	1	5	< 0.1s

Table 1: Experimental results. (1) coordinated, (2) number of rounds per phase, (3) number of invariants provided by the user (safety + termination), (4) number of verification conditions, (5) total solving time.

rounds are grouped together into a phase. This is done, because the computation transition is different for each round within a phase. We verified agreement, validity, irrevocability, and termination.

The *Uniform Voting* algorithm is a deterministic version of the Ben-Or randomized algorithm [3]. The condition for safety is that all environment transitions satisfy the topology predicate $\forall i, j. |HO(i) \cap HO(j)| \geq 1$, called *no-split*. Intuitively, a process decides a value v if all the messages it has received in a specific round are v. Thus two processes decide on different values only if their HO-sets are disjoint. Roughly, the semi-decision procedure succeeds in finding a contradiction, by exploring the explicit non-empty intersection of HO-sets defined by the topology predicate; more specifically the non-empty intersection of HO-sets of two processes that are supposed to decide differently.

Coordinated Uniform Voting and *simplified Coordinated Uniform Voting* [10] are coordinated algorithms. Reasoning about topology predicates similar to *Uniform Voting* leads to safety of these two algorithms. In fact *simplified Coordinated Uniform Voting* is based on an even stronger topology predicate than no-split.

The *One Third Rule*algorithm is our running example. It is designed to be safe without any topology predicate. In this case, the computation transitions enforce that if a processes decides, a majority of processes are in a specific state. In Sec. 5.2, the overview explains how the semi-decision procedure derives a contradiction to prove one of the verification conditions for safety using the majority argument. Our proof of termination is based on a stronger communication predicate than the one provided in [10], namely, it requires two uniform rounds where more that 2/3 of the messages are received by each process.

The *Last Voting* algorithm is an encoding of Paxos [17] in the HO-model. This algorithm is coordinated. Similar to One Third Rule it is safe without any topology predicate, and the algorithm imposes cardinality constraints that create majority sets: Before voting or deciding, the coordinator makes sure that a majority of process acknowledged its messages, such that a decision on v implies $|\{p \,|\, \mathtt{x}(p) = v\}| > n/2$.

The $\mathcal{A}_{T,E}$ algorithm [4] is a generalization of the One Third Rule to value faults that uses different thresholds. It tolerates less than $n/4$ corrupted messages per process. Safety and termination of the algorithm follows the same type of reasoning as the One Third Rule algorithm but require more complex reasoning about the messages. To model value faults, the HO-sets are partitioned into a safe part (SHO) and an altered part (AHO). A message from process p to process q is discarded if $p \notin HO(q)$, delivered as expected if $p \in SHO(q)$, and if $p \in AHO(q)$ an arbitrary message is delivered instead of the original one. The $\mathcal{U}_{T,E}$ algorithm [4] is an consensus algorithm with value faults designed for communication which is live but not safe. For $\mathcal{A}_{T,E}$ and $\mathcal{U}_{T,E}$, to simplify the manually encoding, we have considered the intersection of up to three sets rather than two as presented in the semi-decision procedure.

The *FloodMin* algorithm is a synchronous consensus algorithm tolerating at most f crash fault [19, Chapter 6.2.1]. In each round the processes sends their value to all the processes and keep the smallest received value. Executing $f + 1$ rounds guarantees that there is at least one round where no process crashes. Agreement is reached in this (special) round. The invariant captures that fact by counting the number of crashed process and relating it to the number of processes with different values, i.e. $|C| < r \Rightarrow \exists v.\, |\{p \mid \mathtt{x}(p) = v\}| = n$. Proving termination requires a ranking function.

7 Related Work

The verification of distributed algorithms is a very challenging task. Indeed, most of the verification problems are undecidable for parameterized systems [2, 23]. Infinite-state model-checking techniques may be applied if one restricts the type of actions performed by the processes. Particular classes of systems which are monotonic enjoy good decidability properties [1, 13]. Fault-tolerant distributed algorithms cannot be modeled as such restricted systems. Recently, John et al. [15] developed abstractions suitable for model-checking threshold-based fault-tolerant distributed algorithms.

Orthogonally to the model-checking approach and closer to our approach is the formalization of programs and their specifications in logics where the satisfiability question is decidable. Very expressive logics have been explored for the verification of data structures and $\mathbb{CL}$ is a new combination of the constructs present in those logics. The array property fragment [6] admits a limited form of quantifier alternation which is close to ours. Reasoning about sets and cardinality constraints is present in BAPA [16]. However, BAPA does not combine well with function symbols over sets [24]. Logics for linked heap structures such as lists are similar to $\mathbb{CL}$ if we encode sets as lists and cardinality constraints as length constraints. STRAND [20] and CSL [5] offer more quantifier alternations and richer constraints on data but have more limited cardinality constraints. Both logics have decision procedures based on a small model property.

If one accepts less automation, distributed algorithms can be formalized and verified in interactive proof assistants. For instance, Isabelle has been used to

verify algorithms in the heard-of model [9]. The verification of distributed systems has also been tackled using the TLA+ specification language [18].

References

1. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.K.: General decidability theorems for infinite-state systems (1996)
2. Apt, K., Kozen, D.: Limits for automatic verification of finite-state concurrent systems. Inf. Process. Lett. 15, 307–309 (1986)
3. Ben-Or, M.: Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In: PODC. pp. 27–30. ACM (1983)
4. Biely, M., Charron-Bost, B., Gaillard, A., Hutle, M., Schiper, A., Widder, J.: Tolerating corrupted communication. In: PODC. pp. 244–253 (2007)
5. Bouajjani, A., Drăgoi, C., Enea, C., Sighireanu, M.: A logic-based framework for reasoning about composite data structures. In: CONCUR. Springer (2009)
6. Bradley, A.R., Manna, Z., Sipma, H.B.: What's decidable about arrays? In: VMCAI. pp. 427–442 (2006)
7. Brasileiro, F., Greve, F., Mostefaoui, A., Raynal, M.: Consensus in one communication step. In: Parallel Computing Technologies. Springer (2001)
8. Burrows, M.: The chubby lock service for loosely-coupled distributed systems. In: OSDI. USENIX Association, Berkeley, CA, USA (2006)
9. Charron-Bost, B., Merz, S.: Formal verification of a consensus algorithm in the heard-of model. Int. J. Software and Informatics 3(2-3), 273–303 (2009)
10. Charron-Bost, B., Schiper, A.: The heard-of model: computing in distributed systems with benign faults. Distributed Computing 22(1), 49–71 (2009)
11. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM 32(2), 374–382 (Apr 1985)
12. Függer, M., Schmid, U.: Reconciling fault-tolerant distributed computing and systems-on-chip. Dist. Comp. 24(6), 323–355 (2012)
13. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. Journal of the ACM 39, 675–735 (1992)
14. Hunt, P., Konar, M., Junqueira, F.P., Reed, B.: Zookeeper: wait-free coordination for internet-scale systems. In: USENIXATC. USENIX Association (2010)
15. John, A., Konnov, I., Schmid, U., Veith, H., Widder, J.: Parameterized model checking of fault-tolerant distributed algorithms by abstraction. In: FMCAD. pp. 201–209 (2013)
16. Kuncak, V., Nguyen, H.H., Rinard, M.C.: An algorithm for deciding BAPA: Boolean algebra with presburger arithmetic. In: CADE. pp. 260–277 (2005)
17. Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. (1998)
18. Lamport, L.: Distributed algorithms in TLA (abstract). In: PODC (2000)
19. Lynch, N.: Distributed Algorithms. Morgan Kaufman (1996)
20. Madhusudan, P., Parlato, G., Qiu, X.: Decidable logics combining heap structures and data. In: POPL. pp. 611–622. ACM (2011)
21. Moura, L., Bjorner, N.: Z3: An efficient SMT solver. In: TACAS, pp. 337–340. Springer Berlin Heidelberg (2008)
22. Santoro, N., Widmayer, P.: Time is not a healer. In: STACS. LNCS, vol. 349, pp. 304–313 (1989)
23. Suzuki, I.: Proving properties of a ring of finite-state machines. Inf. Process. Lett. 28(4), 213–214 (Jul 1988)
24. Yessenov, K., Piskac, R., Kuncak, V.: Collections, cardinalities, and relations. In: VMCAI. pp. 380–395 (2010)