Paxos Made EPR: Decidable Reasoning about Distributed Protocols

ODED PADON, Tel Aviv University, Israel
GIULIANO LOSA, University of California, Los Angeles, USA
MOOLY SAGIV, Tel Aviv University, Israel
SHARON SHOHAM, Tel Aviv University, Israel

Distributed protocols such as Paxos play an important role in many computer systems. Therefore, a bug in a distributed protocol may have tremendous effects. Accordingly, a lot of effort has been invested in verifying such protocols. However, checking invariants of such protocols is undecidable and hard in practice, as it requires reasoning about an unbounded number of nodes and messages. Moreover, protocol actions and invariants involve both quantifier alternations and higher-order concepts such as set cardinalities and arithmetic.

This paper makes a step towards automatic verification of such protocols. We aim at a technique that can verify correct protocols and identify bugs in incorrect protocols. To this end, we develop a methodology for deductive verification based on effectively propositional logic (EPR)—a decidable fragment of first-order logic (also known as the Bernays-Schönfinkel-Ramsey class). In addition to decidability, EPR also enjoys the finite model property, allowing to display violations as finite structures which are intuitive for users. Our methodology involves modeling protocols using general (uninterpreted) first-order logic, and then systematically transforming the model to obtain a model and an inductive invariant that are decidable to check. The steps of the transformations are also mechanically checked, ensuring the soundness of the method. We have used our methodology to verify the safety of Paxos, and several of its variants, including Multi-Paxos, Vertical Paxos, Fast Paxos, Flexible Paxos and Stoppable Paxos. To the best of our knowledge, this work is the first to verify these protocols using a decidable logic, and the first formal verification of Vertical Paxos, Fast Paxos and Stoppable Paxos.

CCS Concepts: • **Software and its engineering → Formal methods**; • **Networks** → *Protocol correctness*; • **Theory of computation** → *Logic and verification*;

Additional Key Words and Phrases: Paxos, safety verification, inductive invariants, deductive verification, effectively propositional logic, distributed systems

ACM Reference Format:
Oded Padon, Giuliano Losa, Mooly Sagiv, and Sharon Shoham. 2017. Paxos Made EPR: Decidable Reasoning about Distributed Protocols. *Proc. ACM Program. Lang.* 1, OOPSLA, Article 108 (October 2017), 31 pages.
https://doi.org/10.1145/3133932

1 INTRODUCTION

Paxos is a family of protocols for solving consensus in a network of unreliable processors with unreliable communication. Consensus is the process of deciding on one result among a group of participants. Paxos protocols play an important role in our daily life. For example, Google uses the Paxos algorithm in their Chubby distributed lock service in order to keep replicas consistent in

2475-1421/2017/10-ART108
https://doi.org/10.1145/3133932

case of failure [Burrows 2006]. VMware uses a Paxos-based protocol within the NSX Controller. Amazon Web Services uses Paxos-like algorithms extensively to power its platform [Newcombe et al. 2015]. The key safety property of Paxos is consistency: processors cannot decide on different values.

Due to its importance, verifying the safety of distributed protocols like Paxos is an ongoing research challenge. The systems and programming languages communities have had several recent success stories in verifying the safety of Paxos-like protocols in projects such as IronFleet [Hawblitzel et al. 2015], Verdi [Wilcox et al. 2015], and PSync [Dragoi et al. 2016][1].

1.1 Main Results

This work aims to increase the level of automation in verification of distributed protocols, hoping that it will eventually lead to wider adoption of formal verification in this domain. We follow IronFleet, Verdi, and PSync, in requiring that the user supplies inductive invariants for the protocols. We aim to automate the process of checking the inductiveness of the user-supplied invariants. The goal is that the system can reliably produce in finite time either a proof that the invariant is inductive or display a comprehensible counterexample to induction (CTI), i.e., a concrete transition of the protocol from state s to state s' such that s satisfies the given invariant and s' does not[2]. Such a task seems very difficult since these protocols are usually expressed in rich programming languages in which automatically checking inductive invariants is both undecidable and very hard in practice. In fact, in the IronFleet project, it was observed that undecidability of the reasoning performed by Z3 [de Moura and Bjørner 2008] is a major hurdle in their verification process.

1.1.1 Criteria for Automatic Deductive Verification. We aim for an automated deductive verification technique that achieves three goals:

Natural Making the invariants readable even for users who are not expert in the tools.

Completeness Making sure that if the invariant is inductive then the solver is guaranteed to prove it.

Finite Counterexamples Guaranteeing that if the invariant is not inductive then the solver can display a concrete counterexample to induction with a finite number of nodes which can be diagnosed by users.

These goals are highly ambitious. Expressing the verification conditions in a decidable logic with a small model property (e.g., EPR [Piskac et al. 2010]) will guarantee Completeness and Finite Counterexamples. However, it is not clear how to model complex protocols like Paxos in such logics. Consensus protocols such as Paxos often require higher-order reasoning about sets of nodes (majority sets or quorums), combined with complex quantification. In fact, some researchers conjectured that decidable logics are too restrictive to be useful.

Furthermore, we are aiming to obtain natural invariants. We decided to verify the designs of the protocols and not their implementations since the invariants are more natural and since we wanted to avoid dealing with low level implementation issues. In the future we plan to use refinement to synthesize efficient low level implementations. Systems such as Alloy [Jackson 2006] and TLA [Lamport 2002] have already been used for finding bugs in protocols and inductive invariants (e.g., by Amazon [Newcombe et al. 2015]). Again they verify and identify faults in the designs and not the actual implementation. However, in contrast to our approach, they cannot automatically produce proofs for inductiveness (Completeness).

[1] IronFleet and PSync also verify certain liveness properties.

[2] Such a CTI indicates that there is a bug in the protocol itself, or that the provided invariant is inadequate (e.g., too weak or too strong).

1.1.2 A Reusable Verification Methodology. In this work, we develop a novel reusable verification methodology based on Effectively Propositional logic (EPR) for achieving the above goals. Our methodology allows the expression of complex protocols and systems, while guaranteeing that the verification conditions are expressed in EPR. EPR provides both decidability and finite counterexamples, and is supported by existing solvers (e.g., Z3 [de Moura and Bjørner 2008], iProver [Korovin 2008], VAMPIRE [Riazanov and Voronkov 2002], CVC4 [Barrett et al. 2011]). We have used our methodology to verify the safety of Paxos, and several of its variants, including Multi-Paxos, Vertical Paxos, Fast Paxos, Flexible Paxos and Stoppable Paxos. To the best of our knowledge, this work is the first to verify these protocols using a decidable logic, and, in the case of Vertical Paxos, Fast Paxos, and Stoppable Paxos, it is also the first mechanized safety proof.

We have also compared our methodology to a traditional approach based on a state-of-the-art interactive theorem prover—Isabelle/HOL [Nipkow et al. 2002]. Our comparison shows that the inductive invariants used are very similar in both approaches (Natural), and that our methodology allows more reliable and predictable automation: an interactive theorem prover can discharge proof obligations to theorem provers using undecidable theories, but these often fail due to the undecidability. In such cases, it requires an experienced expert user to prove the inductive invariant. In contrast, with our methodology all the verification conditions are decidable and therefore checking them is fully automated.

First-order uninterpreted abstraction. The first phase in our verification process is expressing the system and invariant in (undecidable) many-sorted first-order logic over uninterpreted structures. This is in contrast to SMT which allows the use of interpreted theories such as arithmetic and the theory of arrays. The use of theories is natural specifically for handling low level aspects such as machine arithmetic and low level storage. However, SMT leads to inherent undecidability with quantifiers which are used to model unbounded systems. In contrast to SMT, we handle concepts, such as arithmetic and set cardinalities, using abstraction expressible in first-order logic, e.g., a totally ordered set instead of the natural numbers. This involves coming up with domain knowledge encoded as first-order axioms (e.g. a first-order formula expressing transitivity of a total order).

We are encouraged by the simplicity of our abstractions and the fact that they are precise enough to prove complex protocols. We also note that using first-order logic has led us to axioms and invariants that elegantly capture the essence of the protocols. This is also enabled by the fact that we are modeling high-level protocols and not their low level implementations.

At the end of this phase, the verification conditions are in general first-order logic. This is already useful as it allows to use resolution-based theorem provers (e.g., SPASS [Weidenbach et al. 2009] and VAMPIRE [Riazanov and Voronkov 2002]). Yet, at this stage the verification conditions are still undecidable, and solvers are not guaranteed to terminate.

One way to obtain decidability is to restrict quantifier alternations. We examine the *quantifier alternation graph* of the verification condition, which connects sorts that alternate in $\forall\exists$ quantification. When this graph contains cycles, solvers such as Z3 often diverge into infinite loops while instantiating quantifiers. This issue is avoided when the graph is acyclic, in which case the verification condition is essentially in EPR. Therefore, the second phase of our methodology provides a systematic way to soundly eliminate the cycles.

Eliminating quantifier alternations using derived relations. The most creative part in our methodology is adding derived relations and rewriting the code to break the cycles in the quantifier alternation graph. The main idea is to capture an existential formula by a derived relation, and then to use the derived relation as a substitute for the formula, both in the code and in the invariant, thus eliminating some quantifier alternations. The user is responsible for defining the derived relations and performing the rewrites. The system automatically generates update code for the

derived relations, and automatically checks the soundness of the rewrites. For the generation of update code, we exploit the locality of updates, as relations (used for defining the derived relations) are updated by inserting a single entry at a time. We identify a class of formulas for which this automatic procedure is always possible and use this class for verifying the Paxos protocols.

We are encouraged by the fact that the transformations needed in this step are reusable across all Paxos variants we consider. Furthermore, the transformations maintain the simplicity and readability of both the code and the inductive invariants.

1.2 Summary of the rest of the paper

In Section 2 we present the technical background on using first-order logic to express transition systems, and on the EPR fragment. We then develop our general methodology for EPR-based verification in Section 3. Section 4 reviews the Paxos consensus algorithm, which is the basis for all Paxos-like protocols. We present our model of the Paxos consensus algorithm as a transition system in first-order logic in Section 5, and continue to verify it using EPR by applying our methodology in Section 6. In Section 7, we describe our verification of Multi-Paxos using EPR. We briefly discuss the verification of Vertical Paxos, Fast Paxos, Flexible Paxos, and Stoppable Paxos in Section 8. More details about the verification of these variants appear in the extended version of this paper [Padon et al. 2017]. In Section 9 we report on our implementation and experimental evaluation. We discuss related work in Section 10, and Section 11 concludes the paper.

2 BACKGROUND: VERIFICATION USING EPR

In this section we present the necessary background on the formalization of transition systems using first-order logic, as well as on the EPR fragment of first-order logic.

2.1 Transition Systems in First Order Logic

We model transition systems using many-sorted first-order logic. We use a vocabulary Σ which consists of sorted constant symbols, function symbols and relation symbols to capture the state of the system, and formulas to capture sets of states and transitions. Formally, given a vocabulary Σ, a *state* is a first-order structure over Σ. We sometimes use *axioms* in the form of closed first-order formulas over Σ, to restrict the set of states to those that satisfy all the axioms. A *transition system* is a pair $(INIT, TR)$, where $INIT$ is the *initial condition* given by a closed formula over Σ, and TR is the *transition relation* given by a closed formula over $\Sigma \uplus \Sigma'$ where Σ is used to describe the source state of the transition and $\Sigma' = \{a' \mid a \in \Sigma\}$ is used to describe the target state. The set of initial states and the set of transitions of the system consist of the states, respectively, pairs of states, that satisfy $INIT$, respectively, TR. We define the set of reachable states of a transition system in the usual way. A *safety property* is expressed by a closed formula P over Σ. The system is *safe* if all of its reachable states satisfy P.

In the paper, we use the *relational modeling language* (RML) [Padon et al. 2016] to express transition systems. An RML program consists of *actions*, each of which consists of a loop-free code that is executed atomically, and corresponds to a single transition. RML commands include non-deterministic choice, sequential composition, and updates to constant symbols, function symbols and relation symbols (representing the system's state), where updates are expressed by first-order formulas. In addition, conditions in RML are expressed using **assume** commands. RML programs naturally translate to formulas $(INIT, TR)$, where TR is a disjunction of the transition relation formulas associated with each action (see [Padon et al. 2016] for details of the translation). As such, we will use models, programs and transition systems interchangeably throughout the paper. We note that RML is Turing-complete, and remains so when $INIT$ and TR are restricted to the EPR fragment.

A closed first-order formula INV over Σ is an *inductive invariant* for a transition system $(INIT, TR)$ if $INIT \models INV$ and $INV \wedge TR \models INV'$, where INV' results from substituting every symbol in INV by its primed version. These requirements ensure that an inductive invariant represents a superset of the reachable states. Given a safety property P, an inductive invariant INV proves that the transition system is safe if $INV \models P$. Equivalently, INV proves safety of $(INIT, TR)$ for P if the following formulas are unsatisfiable: (i) $INIT \wedge \neg INV$, (ii) $INV \wedge TR \wedge \neg INV'$, and (iii) $INV \wedge \neg P$. We refer to these formulas as the *verification condition* of INV. When $INV \wedge TR \wedge \neg INV'$ is satisfiable, and $(s, s') \models INV \wedge TR \wedge \neg INV'$, we say that the transition (s, s') is a *counterexample to induction* (CTI).

2.2 Extended Effectively Propositional Logic (EPR)

The effectively-propositional (EPR) fragment of first-order logic, also known as the Bernays-Schönfinkel-Ramsey class is restricted to relational first-order formulas (i.e., formulas over a vocabulary that contains constant symbols and relation symbols but no function symbols) with a quantifier prefix $\exists^*\forall^*$ in prenex normal form. Satisfiability of EPR formulas is decidable [Lewis 1980]. Moreover, formulas in this fragment enjoy the *finite model property*, meaning that a satisfiable formula is guaranteed to have a finite model. The size of this model is bounded by the total number of existential quantifiers and constants in the formula. The reason for this is that given an $\exists^*\forall^*$-formula, we can obtain an equi-satisfiable quantifier-free formula by Skolemization, i.e., replacing the existentially quantified variables by constants, and then instantiating the universal quantifiers for all constants. While EPR does not allow any function symbols nor quantifier alternation except $\exists^*\forall^*$, it can be easily extended to allow *stratified* function symbols and quantifier alternation (as formalized below). The extension maintains both the finite model property and the decidability of the satisfiability problem.

The quantifier alternation graph. Let φ be a formula in negation normal form over a many-sorted signature Σ with a set of sorts $\mathcal{S}$. We define the *quantifier alternation graph* of φ as a directed graph where the set of vertices is the set of sorts, $\mathcal{S}$, and the set of directed edges, called $\forall\exists$ edges, is defined as follows.

- **Function edges:** let f be a function in φ from sorts $s_1, \ldots, s_k$ to sort s. Then there is a $\forall\exists$ edge from s_i to s for every $1 \leq i \leq k$.
- **Quantifier edges:** let $\exists x : s$ be an existential quantifier that resides in the scope of the universal quantifiers $\forall x_1 : s_1, \ldots, \forall x_k : s_k$ in φ. Then there is a $\forall\exists$ edge from s_i to s for every $1 \leq i \leq k$.

Intuitively, the quantifier edges correspond to the edges that would arise as function edges if Skolemization is applied.

Extended EPR. A formula φ is *stratified* if its quantifier alternation graph is acyclic. The *extended EPR* fragment consists of all stratified formulas. This fragment maintains the finite model property and the decidability of EPR. The reason for this is that, after Skolemization, the vocabulary of a stratified formula can only generate a finite set of ground terms. This allows complete instantiation of the universal quantifiers in the Skolemized formula, as in EPR. In the sequel, whenever we say a formula is in EPR, we refer to the extended EPR fragment.

3 METHODOLOGY FOR DECIDABLE VERIFICATION

In this section we explain the general methodology that we follow in our efforts to verify Paxos using decidable reasoning. While this paper focuses on Paxos and its variant, the methodology is more general and can be useful for verifying other systems as well.

3.1 Modeling in Uninterpreted First-Order Logic

The first step in our verification methodology is to express the protocol as a transition system in many-sorted uninterpreted first-order logic. This step involves some abstraction, since protocols usually employ concepts that are not directly expressible in uninterpreted first-order logic.

3.1.1 Axiomatizing Interpreted Domains. One of the challenges we face is modeling an interpreted domain using *uninterpreted* first-order logic. Distributed algorithms often use values from interpreted domains, the most common example being the natural numbers. These domains are usually not precisely expressible in uninterpreted first order logic.

To express an interpreted domain, such as the natural numbers, in uninterpreted first-order logic, we add a sort that represents elements of the interpreted domain, and uninterpreted symbols to represent the interpreted symbols (e.g. a $\leq$ binary relation). We capture *part* of the intended interpretation of the symbols by introducing *axioms* to the model. The axioms are a finite set of first-order logic formulas that are valid in the interpreted domain. By adding them to the model, we allow the proof of verification conditions to rely on these axioms. By using only axioms that are valid in the interpreted domain, we guarantee that any invariant proved for the first-order model is also valid for the actual system.

One important example for axioms expressible in first-order logic is the axiomatization of total orders. In many cases, natural numbers are used as a way to enforce a total order on a set of elements. In such cases, we can add a binary relation $\leq$, along with the axioms listed in Fig. 1, which precisely capture the properties of a total order.

$$\forall x.\ x \leq x$$
$$\forall x, y, z.\ x \leq y \wedge y \leq z \rightarrow x \leq z$$
$$\forall x, y.\ x \leq y \wedge y \leq x \rightarrow x = y$$
$$\forall x, y.\ x \leq y \vee y \leq x$$

Fig. 1. Axiomatization of total order.

3.1.2 Expressing Higher-Order Logic. Another hurdle to using first-order logic is the fact that algorithms and their invariants often use sets and functions as first class values, e.g. by quantifying over them, sending them in a message, etc. Consider an algorithm in which messages contain a set of nodes as one of the message fields. Then, the set of messages sent so far (which may be part of the state of the system) is a set of tuples, where one of the elements in the tuples is itself a set of nodes. Similarly, messages may contain maps, which are naturally modeled by functions (e.g., a message may contain a map from nodes to values). In such cases, the invariants needed to prove the algorithms will usually include higher-order quantification.

While higher-order logic cannot be fully reduced to first-order logic, it is well-known that we can partly express high-order concepts in first-order logic in the following way.

Sets. Suppose we want to express quantification over sets of nodes. We add a new sort called nodeset, and a binary relation $member : \text{node}, \text{nodeset}$. We then use $member(n, s)$ instead of $n \in s$, and express quantification over sets of nodes as quantification over nodeset. Typically, we will need to add first-order assumptions or axioms to correctly express the algorithm and to prove its inductive invariant. For example, the algorithm may set s to the empty set as part of a transition. We can translate this in the transition relation using $\forall x : \text{node}.\ \neg member(x, s')$ (where s' is the value of s after the transition).

Functions. Functions can be encoded as first-order elements in a similar way. Suppose messages in the algorithm contain a map from nodes to values. In this case, we can add a new first-order sort called map, and a function symbol $apply : \text{map}, \text{node} \rightarrow \text{value}$. Then, we can use $apply(m, n)$ instead of $m(n)$, and replace quantification over functions with quantification of the first-order sort map. As before, we may need to add axioms that capture some of the intended second-order meaning of the sort map.

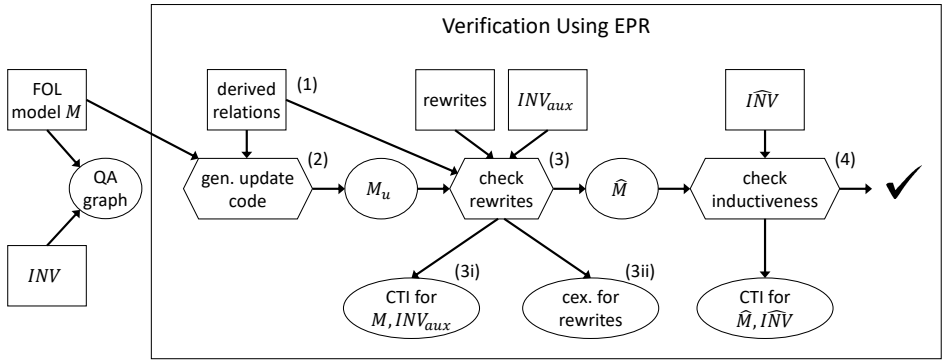

Fig. 2. Flow chart of the methodology for verification using EPR. User provided inputs are depicted as rectangles, automated procedures are depicted as hexagons, and automatically generated outputs are depicted as ellipses. The original first-order logic model $M = (INIT, TR)$, and the original (first-order logic) inductive invariant INV result in a quantifier alternation graph, which guides the process. The transformation to EPR is carried in steps 1-4, detailed in Section 3.2. In step 1, the user provides definitions for derived relations. In step 2, update code is automatically generated, resulting in $M_u = (INIT_u, TR_u)$. In step 3, the user provides rewrites that use the derived relations, as well as an auxiliary inductive invariant to prove the soundness of the rewrites. An automated procedure first checks the soundness of the rewrites (and provides a counterexample in case they are unsound, or a counterexample to induction if the auxiliary inductive invariant is not inductive), and then outputs the transformed model $\widehat{M} = (\widehat{INIT}, \widehat{TR})$. In step 4, the user provides an inductive invariant $\widehat{INV}$ that proves safety of the transformed model, and an automated check either verifies the inductiveness or provides a counterexample to induction.

While this encoding is sound (as long as we only use axioms that are valid in the higher-order interpretation), it cannot be made complete due to the limitation of first-order logic. However, we did not experience this incompleteness to be a practical hurdle for verification in first-order logic.

3.1.3 Semi-Bounded Verification. Given a transition system in first-order logic with a candidate inductive invariant, it may still be undecidable to check the resulting verification condition. However, bounded verification is decidable, and extremely useful for debugging the model before continuing with the efforts of unbounded verification. Contrary to the usual practice of bounding the number of elements in each sort for bounded verification, we use the quantifier alternation graph to determine only a *subset* of the sorts to bound in order to make verification decidable. We call this procedure *semi-bounded verification*, and it follows from the observation that whenever we make a sort bounded, we can remove its node from the quantifier alternation graph. When the resulting graph becomes acyclic, satisfiability is decidable without bounding the sizes of the remaining sorts.

3.2 Transformation to EPR Using Derived Relations

The second step in our methodology for decidable *unbounded* verification is to transform the model expressed in first-order logic to a model that has an inductive invariant whose verification condition is in EPR, and is therefore decidable to check. The methodology is manual, but following it ensures soundness of the verification process. The key idea is to use *derived relations* to simplify the transition relation and the inductive invariant. Derived relations extend the state of the system and are updated in its transitions. Derived relations are somewhat analogous to ghost variables. However there are two key differences. First, derived relations are typically not used to record the history of an execution. Instead, they capture properties of the current state in a way that facilitates

verification using EPR. Second, derived relations are not only updated in the transitions, but can also affect them.

The transformation of the model using derived relations is conducted in steps, as detailed below. The various steps are depicted in Fig. 2. The inputs provided by the user are depicted by rectangles, while the automated procedures are depicted as hexagons, and their outputs are depicted as ellipses. As illustrated by the figure, the user is guided by the quantifier alternation graph of the verification conditions.

In the sequel, we fix a model over a vocabulary Σ and let *INIT* and *TR* denote its initial condition and transition relation, respectively.

(1) Defining a derived relation. In the first, and most creative part of the process, the user identifies an existentially quantified formula $\psi(\bar{x})$ that will be captured by a derived relation $r(\bar{x})$. The selection of ψ is guided by the quantifier alternation graph of the verification condition, with the purpose of eliminating cycles it contains. Quantifier alternations in the verification condition originate both from the model and the inductive invariant. As we shall see, using r will allow us to eliminate some quantifier alternations. As an example for demonstrating the next steps, consider a program defined with a binary relation p, and let $r(x)$ be a derived relation capturing the formula $\psi(x) = \exists y.\, p(x, y)$.

(2) Tracking ψ by r. This step automatically extends the model into a model $(INIT_u, TR_u)$ over vocabulary $\widehat{\Sigma} = \Sigma \cup \{r\}$ which makes the same transitions as before, but also updates r to capture ψ. Formally, the transformed model $(INIT_u, TR_u)$ over $\widehat{\Sigma}$ is obtained by adding: (i) an initial condition that initializes r, and (ii) *update code* that modifies r whenever the relations mentioned in ψ are modified. The initial condition and update code are automatically generated in a way that guarantees that the following formula is an invariant of $(INIT_u, TR_u)$:

$$\forall \bar{x}.\, r(\bar{x}) \leftrightarrow \psi(\bar{x}). \tag{1}$$

We call this invariant the *representation invariant* of r. Our scheme for automatically obtaining $(INIT_u, TR_u)$ and the class of formulas ψ that it supports, are discussed in Section 3.3. In our example, suppose that initially p is empty. Then, the resulting model would initialize r to be empty as well. For an action that inserts a pair (a, b) to p, the resulting model would contain update code that inserts a to r.

(3) Rewriting the transitions using r. In this step, the user exploits r to eliminate quantifier alternations in the verification condition by rewriting the system's transitions, obtaining a model $(\widehat{INIT}, \widehat{TR})$ defined over $\widehat{\Sigma}$. The idea is to rewrite the transitions in a way that ensures that the reachable states are unchanged, while eliminating quantifier alternations. This is done by rewriting some program conditions used in **assume** commands in the code (e.g., to use r instead of ψ, but other rewrites are also possible). The vocabulary of the model does not change further in this step, nor does the initial condition (i.e., $\widehat{INIT} = INIT_u$).

While the rewrites are performed by the user, we automatically check that the effect of the modified commands on the reachable states remains the same (under the assumption of the representation invariant). Suppose the user rewrites **assume** φ to **assume** $\widehat{\varphi}$. The simplest way to ensure this has the same effect on the reachable states is to check that the following *rewrite condition* is valid: $\varphi \leftrightarrow \widehat{\varphi}\,[\psi(\bar{x}) \,/\, r(\bar{x})]$. This condition guarantees that the two formulas φ and $\widehat{\varphi}$ are equivalent in any reachable state, due to the representation invariant. In some cases, the rewrite is such that $\widehat{\varphi}\,[\psi(\bar{x}) \,/\, r(\bar{x})]$ is syntactically identical to φ, which makes the rewrite condition trivial.

However, to allow greater flexibility in rewriting the code, we allow using an EPR check to verify the rewrite condition, and also relax the condition given above in two ways. First, we observe that it suffices to verify the equivalence of subformulas of φ that were modified by the rewrite.

Formally, if φ is syntactically identical to $\widehat{\varphi}\left[\theta_1(\bar{y}_1) \,/\, \widehat{\theta}_1(\bar{y}_1), \ldots, \theta_k(\bar{y}_k) \,/\, \widehat{\theta}_k(\bar{y}_k)\right]$, then to establish the rewrite condition, it suffices to prove that for every $1 \le i \le k$ the following equivalence is valid: $\theta_i \leftrightarrow \widehat{\theta}_i\,[\psi(\bar{x}) \,/\, r(\bar{x})]$. (The case where φ was completely modified is captured by the case where $k = 1$, $\varphi = \theta_1$ and $\widehat{\varphi} = \widehat{\theta}_1$.) Second, and more importantly, recall that we are only interested in preserving the transitions from reachable states of the system. Thus, we allow the user to provide an auxiliary invariant $\mathit{INV}_{\text{aux}}$ (by default $\mathit{INV}_{\text{aux}} = \mathit{true}$) which is used to prove that the reachable transitions remain unchanged after the transformation. Technically, this is done by automatically checking that

(i) $\mathit{INV}_{\text{aux}}$ is an inductive invariant of $(\mathit{INIT}, \mathit{TR})$, and
(ii) the following rewrite condition holds for every $1 \le i \le k$:

$$\mathit{INV}_{\text{aux}} \wedge g \models \theta_i \leftrightarrow \widehat{\theta}_i\,[\psi(\bar{x}) \,/\, r(\bar{x})], \tag{2}$$

where g captures additional conditions that guard the modified **assume** command (g is automatically computed from the program).

These conditions guarantee that the two formulas φ and $\widehat{\varphi}$ are equivalent whenever the modified **assume** command is executed. To ensure that these checks can be done automatically, we require that the corresponding formulas are in EPR. We note that verifying $\mathit{INV}_{\text{aux}}$ for $(\mathit{INIT}, \mathit{TR})$ can be possible in EPR even in cases where verifying safety of $(\mathit{INIT}, \mathit{TR})$ is not in EPR, since $\mathit{INV}_{\text{aux}}$ can be weaker (and contain less quantifier alternations) than an invariant that proves safety.

In our example, suppose the program contains the command **assume** $\exists y.\, p(a,y)$. Then we could rewrite it to **assume** $r(a)$. For a more sophisticated example, suppose that the program contains the command **assume** $\exists y.\, p(a,y) \wedge q(a,y)$, and suppose this command is guarded by the condition $u(a)$ (i.e., the **assume** only happens if $u(a)$ holds). Suppose further that we can verify that $\mathit{INV}_{\text{aux}} = \forall x, y.\, u(x) \wedge p(x,y) \rightarrow q(x,y)$ is an invariant of the original system. Then we could rewrite the assume command as **assume** $r(a)$ since $(\forall x, y.\, u(x) \wedge p(x,y) \rightarrow q(x,y)) \wedge u(a) \models (\exists y.\, p(a,y) \wedge q(a,y)) \leftrightarrow \exists y.\, p(a,y)$.

(4) Providing an inductive invariant. Finally, the user proves the safety of the transformed model $(\widehat{\mathit{INIT}}, \widehat{\mathit{TR}})$ by providing an inductive invariant $\widehat{\mathit{INV}}$ for it, whose verification condition will be in EPR. Usually this is composed of: (i) Using r in the inductive invariant as a substitute to using ψ. The point here is that using ψ would introduce quantifier alternations, and using r instead avoids them. In our example, the safety proof might require the property that $\forall x.\, u(x) \rightarrow \exists y.\, p(x,y)$, and using r we can express this as $\forall x.\, u(x) \rightarrow r(x)$. (ii) Letting the inductive invariant express some properties that are implied by the representation invariant. Note that expressing the full representation invariant would typically introduce quantifier alternations that break stratification. However, some properties implied by it may still be expressible while keeping the verification condition in EPR. In our example, we may add $\forall x, y.\, p(x,y) \rightarrow r(x)$ to the inductive invariant. Note that adding $\forall x.\, r(x) \rightarrow \exists y.\, p(x,y)$ to the inductive invariant would make the verification condition outside of EPR.

Given $(\widehat{\mathit{INIT}}, \widehat{\mathit{TR}})$ and $\widehat{\mathit{INV}}$, we can now automatically derive the verification conditions in EPR and check that they hold. The following theorem summarizes the soundness of the approach:

Theorem 3.1 (Soundness). *Let $(\mathit{INIT}, \mathit{TR})$ be a model over vocabulary Σ, and P be a safety property over Σ. If $(\widehat{\mathit{INIT}}, \widehat{\mathit{TR}})$ is a model obtained by the above procedure, and $\widehat{\mathit{INV}}$ is an inductive invariant for it such that $\widehat{\mathit{INV}} \models P$, then P holds in all reachable states of $(\mathit{INIT}, \mathit{TR})$.*

Proof Sketch. Let $B = \{(s, \hat{s}) \mid s \in R \wedge \hat{s} \in \hat{R} \wedge \hat{s}|_\Sigma = s\}$, where R and $\hat{R}$ denote the reachable states of $(\mathit{INIT}, \mathit{TR})$ and $(\widehat{\mathit{INIT}}, \widehat{\mathit{TR}})$ respectively, and $\hat{s}|_\Sigma$ denotes the projection of a state $\hat{s}$ (defined

over $\widehat{\Sigma}$) to Σ. Steps 2 and 3 of the transformation above ensure that B is a bisimulation relation between $(INIT, TR)$ and $(\widehat{INIT}, \widehat{TR})$, i.e., every transition possible in the reachable states of one of these systems has a corresponding transition in the other. This ensures that $(\widehat{INIT}, \widehat{TR})$ has the same reachable states as $(INIT, TR)$, up to the addition of relation r. Therefore, any safety property expressed over Σ which is verified to hold in $(\widehat{INIT}, \widehat{TR})$ also holds in $(INIT, TR)$. □

As shown in the proof of Theorem 3.1, the transformed model $(\widehat{INIT}, \widehat{TR})$ is bisimilar to the original model. While this ensures that both are equivalent w.r.t. to the safety property, note that we check safety by checking inductiveness of a candidate invariant. Unlike safety, inductiveness is not necessarily preserved by the transformation. Namely, given a candidate inductive invariant $\widehat{INV}$ which is not inductive for $(\widehat{INIT}, \widehat{TR})$, the counterexample to induction cannot in general be transformed to the original model, as it might depend on the derived relations and the rewritten **assume** commands. An example of this phenomenon appears in Section 6.2.

Using the methodology. Our description above explains a final successful verification using the proposed methodology. As always, obtaining this involves a series of attempts, where in each attempt the user provides the verification inputs, and gets a counterexample. Each counterexample guides the user to modify the verification inputs, until eventually verification is achieved. As depicted in Fig. 2, with the EPR verification methodology, the user provides 5 inputs, and could obtain 3 kinds of counterexamples. The inputs are the model, the derived relations, the rewrites, the auxiliary invariant for proving the soundness of the rewrites, and finally the inductive invariant for the resulting model. The possible counterexamples are either a counterexample to inductiveness (CTI) for the auxiliary invariant and the original model, or a counterexample to the soundness of the rewrite itself, or a CTI for the inductive invariant of the transformed model. After obtaining any of the 3 kinds of counterexamples, the user can modify any one of the 5 inputs. For example, a CTI for the inductive invariant of the transformed model may be eliminated by changing the inductive invariant itself, but it may also be overcome by an additional rewrite, which in turn requires an auxiliary invariant for its soundness proof. Indeed, we shall see an example of this in Section 6.2.

The task of managing the inter-dependence between the 5 verification inputs may seem daunting, and indeed it requires some expertise and creativity from the user. This is expected, since the inputs from the user reduce the undecidable problem of safety verification to decidable EPR checks. This burden on the user is eased by the fact that for every input, the user always obtains an answer from the system, either in the form of successful verification, or in the form of a *finite* counterexample, which is displayed graphically and guides the user towards the solution. Furthermore, our experience shows that most of the creative effort is reusable across similar protocols. In the verification of all the variants of Paxos we consider in this work, we use the same two derived relations and very similar rewrites (as explained in Sections 6 and 8).

Incompleteness of EPR verification. While the transformation using a given set of derived relations and rewrites results in a bisimilar transition system, the methodology for EPR verification is not complete. This is expected, as there can be no complete proof system for safety in a formalism that is Turing-complete. For the EPR verification methodology, the incompleteness can arise from several sources. It may happen that after applying the transformation, the resulting transition system, while safe, cannot be verified with an inductive invariant that results in EPR verification conditions. Another potential source for incompleteness is our requirement that the rewrites should also be verified in EPR. It can be the case that a certain (sound) rewrite leads to a system that can be verified using EPR, but the soundness of the rewrite itself cannot be verified using EPR. Another potential source of incompleteness can be the inability to express sufficiently powerful axioms about the underlying domain. We note that the three mentioned issues interact with each other, as

it may be the case that a certain axiom is expressible in first-order logic, but it happens to introduce a quantifier alternation cycle, when considered together with either the inductive invariant or the verification conditions for the rewrites.

We consider developing a proof-theoretic understanding of which systems can and cannot be verified using EPR to be an intriguing direction for future investigation. We are encouraged by the fact that in practice, the proposed methodology has proven itself to be powerful enough to verify Paxos and its variants considered in this work.

Multiple derived relations. For simplicity, the description above considered a single derived relation. In practice, we usually add multiple derived relations, where each one captures a different formula. The methodology remains the same, and each derived relation allows us to transform the model and eliminate more quantifier alternations, until the resulting model can be verified in EPR. In this case, the resulting inductive invariant may include properties implied by the representation invariants of several relations and relate them directly. For example, suppose we add the following derived relations: $r_1(x)$ defined by $\psi_1(x) = \exists y.\, p(x,y)$, and $r_2(x)$ defined by $\psi_2(x) = \exists y,z.\, p(x,y) \wedge p(y,z)$. Then, the inductive invariant may include the property: $\forall x.\, r_2(x) \rightarrow r_1(x)$.

Overapproximating the reachable states. Our methodology ensures that the transformed model is bisimilar to the original model. It is possible to generalize our methodology and only require that the modified model simulates the original model, which maintains soundness. This may allow more flexibility both in the update code and in the manual rewrites performed by the user.

3.3 Automatic Generation of Update Code

In this subsection, we describe a rather naïve scheme for automatic generation of initial condition and update code for derived relations, which suffices for verification of the Paxos variants considered in this paper. We refer the reader to, e.g., [Paige and Koenig 1982; Reps et al. 2010], for more advanced techniques for generation of update code for derived relations.

We limit the formula $\psi(\bar{x})$ which defines a derived relation $r(\bar{x})$ to have the following form:

$$\psi(x_1,\ldots,x_n) = \exists y_1,\ldots,y_m.\, \varphi(x_1,\ldots,x_n,y_1,\ldots,y_m) \wedge p(x_{i_1},\ldots,x_{i_k},y_1,\ldots,y_m)$$

where φ is a quantifier-free formula, p is a relation symbol and $x_{i_j} \in \{x_1,\ldots,x_n\}$ for every $1 \le j \le k$. Note that p occurs positively, and that it depends on *some* (possibly none) of the variables x_i and *all* of the variables y_i. Our scheme further requires that the relations appearing in φ are never modified, and that p is initially empty and only updated by inserting a single tuple at a time[3].

Since p is initially empty, the initial condition for $r(\bar{x})$ is that it is empty as well, i.e.:

$$\forall x_1,\ldots,x_n.\, \neg r(x_1,\ldots,x_n).$$

The only updates allowed for p are insertions of a single tuple by a command of the form:

$$p(x_{i_1},\ldots,x_{i_k},y_1,\ldots,y_m) := p(x_{i_1},\ldots,x_{i_k},y_1,\ldots,y_m) \vee \left(\bigwedge_{j=1}^{k} x_{i_j} = a_j \wedge \bigwedge_{j=1}^{m} y_j = b_j \right).$$

For such an update, we generate the following update for $r(\bar{x})$:

$$r(x_1,\ldots,x_n) := r(x_1,\ldots,x_n) \vee \left(\varphi(x_1,\ldots,x_n,b_1,\ldots,b_m) \wedge \bigwedge_{j=1}^{k} x_{i_j} = a_j \right).$$

[3]These restrictions can be relaxed, e.g., to support removal of a single tuple or addition of multiple tuples. However, such updates were not needed for verification of the protocols considered in this paper, so for simplicity of the presentation we do not handle them.

Notice that the update code translates to a purely universally quantified formula, since φ is quantifier-free, so it does not introduce any quantifier alternations.

Lemma 3.2. *The above scheme results in a model which maintains the representation invariant:* $\forall \bar{x}.\, r(\bar{x}) \leftrightarrow \psi(\bar{x})$.

Proof Sketch. The representation invariant is an inductive invariant of the resulting model. Initiation is trivial, since both p and r are initially empty. Consecution follows from the following, which is valid in first-order logic: $(\forall \bar{x}.r(\bar{x}) \leftrightarrow \psi(\bar{x})) \wedge \left(\forall \bar{w}, \bar{y}.p'(\bar{w}, \bar{y}) \leftrightarrow p(\bar{w}, \bar{y}) \vee (\bar{w} = \bar{a} \wedge \bar{y} = \bar{b})\right) \wedge \left(\forall \bar{x}.r'(\bar{x}) \leftrightarrow r(\bar{x}) \vee \left(\varphi(\bar{x}, \bar{b}) \wedge \bigwedge_{j=1}^{k} x_{i_j} = a_j\right)\right) \models (\forall \bar{x}.r'(\bar{x}) \leftrightarrow \psi'(\bar{x}))$. □

4 INTRODUCTION TO PAXOS

A popular approach for implementing distributed systems is state-machine replication (SMR) [Schneider 1990], where a (virtual) centralized sequential state machine is replicated across many nodes (processors), providing fault-tolerance and exposing to its clients the familiar semantics of a centralized state machine. SMR can be thought of as repeatedly agreeing on a command to be executed next by the state machine, where each time agreement is obtained by solving a *consensus* problem. In the consensus problem, a set of nodes each propose a value and then reach agreement on a single proposal.

The Paxos family of protocols is widely used in practice for implementing SMR. Its core is the Paxos consensus algorithm [Lamport 1998, 2001]. A Paxos-based SMR implementation executes a sequence of Paxos consensus instances, with various optimizations. The rest of this section explains the Paxos consensus algorithm (whose verification in EPR we discuss in Sections 5 and 6). We return to the broader context of SMR in Section 8.

Setting. We consider a fixed set of nodes, which operate asynchronously and communicate by message passing, where every node can send a message to every node. Messages can be lost, duplicated, and reordered, but they are never corrupted. Nodes can fail by stopping, but otherwise faithfully execute their algorithm. A stop failure of a node can be captured by a loss of all messages to and from this node. Nodes must solve the consensus problem: each node has a value to propose and all nodes must eventually decide on a unique value among the proposals.

Paxos consensus algorithm. We assume that nodes in the Paxos consensus algorithm can all propose values, vote for values, and learn about decisions. The algorithm operates in a sequence of numbered rounds in which nodes can participate. At any given time, different nodes may operate in different rounds, and a node stops participating in a round once it started participating in a higher round. Each round is associated with a single node that is the *owner* of that round. This association from rounds to nodes is static and known to all nodes.

Every round represents a possibility for its owner to propose a value to the other nodes and get it decided on by having a *quorum* of the nodes vote for it in the round. Quorums are sets of nodes such that any two quorums intersect (e.g., sets consisting of a strict majority of the nodes). To avoid the algorithm being blocked by the stop failure of a node which made a proposal, any node can start one of its rounds and make a new proposal in it at any time (in particular, when other rounds are still active) by executing the following two phases:

phase 1. The owner p of round r starts the round by communicating with the other nodes to have a majority of them *join* round r, and to determine which values are *choosable* in lower rounds than r, i.e., values that might have or can still be decided in rounds lower than r.

phase 2. If a value v is choosable in $r' < r$, in order not to contradict a potential decision in r', node p proposes v in round r. If no value is choosable in any $r' < r$, then p proposes a

value of its choice in round r. If a majority of nodes *vote* in round r for p's proposal, then it becomes *decided*.

Note that it is possible for different values to be proposed in different rounds, and also for several decisions to be made in different rounds. Safety is guaranteed by the fact that (by definition of choosable) a value can be decided in a round $r' < r$ only if it is choosable in r', and that if a value v is choosable in round $r' < r$, then a node proposing in r will only propose v. The latter relies on the property that choosable values from prior rounds cannot be missed. Next, we describe in more detail what messages the nodes exchange and how a node makes sure not to miss any choosable value from prior rounds.

Phase 1a: The owner p of round r sends a "start-round" message, requesting all nodes to join round r.

Phase 1b: Upon receiving a request to join round r, a node will only join if it has not yet joined a higher round. If it agrees to join, it will respond with a "join-acknowledgment" message that will also contain its maximal vote so far, i.e., its vote in the highest round prior to r, or $\perp$ if no such vote exists. By sending the join-acknowledgment message, the node promises that it will not join or vote in any round smaller than r.

Phase 2a: After p receives join-acknowledgment messages from a quorum of the nodes, it proposes a value v for round r by sending a "propose" message to all nodes. Node p selects the value v by taking the maximal vote reported by the nodes in their join-acknowledgment messages, i.e., the value that was voted for in the highest round prior to r by any of the nodes whose join-acknowledgment messages formed the quorum. As we will see, only this value can be choosable in any $r' < r$ out of all proposals from lower rounds. If all of these nodes report they have not voted in any prior round, then p may propose any value.

Phase 2b: Upon receiving a propose message proposing value v for round r, a node will ignore it if it already joined a round higher than r, and otherwise it will vote for it, by sending a vote message to all nodes. Whenever a quorum of nodes vote for a value in some round, this value is considered to be decided. Nodes learn this by observing the vote messages.

Note that a node can successfully start a new round or get a value decided only if at least one quorum of nodes is responsive. When quorums are taken to be sets consisting of a strict majority of the nodes, this means Paxos tolerates the failure of at most $\lfloor n/2 \rfloor$ nodes, where n is the total number of nodes. Moreover, Paxos may be caught in a live-lock if nodes keep starting new rounds before any value has a chance to be decided on.

5 PAXOS IN FIRST-ORDER LOGIC

The first step of our verification methodology is to model the Paxos consensus algorithm as a transition system in many-sorted first-order logic over uninterpreted domains. This section explains our model, listed in Fig. 3, as well as its safety proof via an inductive invariant.

5.1 Model of the Protocol

Our model of Paxos involves some abstraction. Since each round r has a unique owner that will exclusively propose in r, we abstract away the owner node and treat the round itself as the proposer. We also abstract the mechanism by which nodes receive the values up for proposal, and allow them to propose arbitrary values.

Additional abstractions are needed as some aspects of the protocol cannot be fully expressed in uninterpreted first-order logic. One such aspect is the fact that round numbers are integers, as arithmetic cannot be fully captured in first-order logic. Another aspect which must be abstracted is

```
1  sort node, quorum, round, value
2
3  relation ≤ : round, round
4  axiom total_order(≤)
5  constant ⊥ : round
6
7  relation member : node, quorum
8  axiom ∀q1, q2 : quorum. ∃n : node. member(n, q1) ∧ member(n, q2)
9
10 relation start_round_msg : round
11 relation join_ack_msg : node, round, round, value
12 relation propose_msg : round, value
13 relation vote_msg : node, round, value
14 relation decision : node, round, value
15
16 init ∀r. ¬start_round_msg(r)
17 init ∀n, r1, r2, v. ¬join_ack_msg(n, r1, r2, v)
18 init ∀r, v. ¬propose_msg(r, v)
19 init ∀n, r, v. ¬vote_msg(n, r, v)
20 init ∀n, r, v. ¬decision(n, r, v)
21
22 action START_ROUND(r : round) {
23   assume r ≠ ⊥
24   start_round_msg(r) := true
25 }
26 action JOIN_ROUND(n : node, r : round) {
27   assume r ≠ ⊥
28   assume start_round_msg(r)
29   assume ¬∃r', r'', v. r' > r ∧ join_ack_msg(n, r', r'', v)
30   # find maximal round in which n voted, and the corresponding vote.
31   # maxr = ⊥ and v is arbitrary when n never voted.
32   local maxr, v := max {(r', v') | vote_msg(n, r', v') ∧ r' < r}
33   join_ack_msg(n, r, maxr, v) := true
34 }
35 action PROPOSE(r : round, q : quorum) {
36   assume r ≠ ⊥
37   assume ∀v. ¬propose_msg(r, v)
38   # 1b from quorum q
39   assume ∀n. member(n, q) → ∃r', v. join_ack_msg(n, r, r', v)
40   # find the maximal round in which a node in the quorum reported
41   # voting, and the corresponding vote.
42   # v is arbitrary if the nodes reported not voting.
43   local maxr, v := max {(r', v') | ∃n. member(n, q)
44                                  ∧join_ack_msg(n, r, r', v') ∧ r' ≠ ⊥}
45   propose_msg(r, v) := true   # propose value v
46 }
47 action VOTE(n : node, r : round, v : value) {
48   assume r ≠ ⊥
49   assume propose_msg(r, v)
50   assume ¬∃r', r'', v. r' > r ∧ join_ack_msg(n, r', r'', v)
51   vote_msg(n, r, v) := true
52 }
53 action LEARN(n : node, r : round, v : value, q : quorum) {
54   assume r ≠ ⊥
55   # 2b from quorum q
56   assume ∀n. member(n, q) → vote_msg(n, r, v)
57   decision(n, r, v) := true
58 }
```

Fig. 3. Model of Paxos consensus algorithm as a transition system in many-sorted first-order logic.

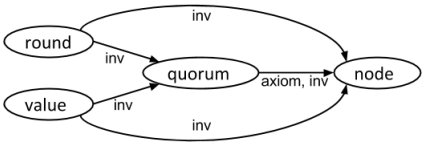

Fig. 4. Quantifier alternation graph for EPR model of Paxos. The graph is obtained for the model of Fig. 5, and the inductive invariant given by the conjunction of eqs. (4) to (10) and (13) to (15). The ∀∃ edges come from: (i) the quorum axiom (Fig. 3 line 8) - edge from quorum to node; (ii) eq. (7) – edges from round and value to quorum, and an edge from quorum to node (from the negation of the inductive invariant in the VC); and (iii) eq. (13) – edges from round, value and quorum to node.

```
1  relation left_round : node, round
2  relation joined_round : node, round
3
4  init ∀n, r. ¬left_round(n, r)
5  init ∀n, r. ¬joined_round(n, r)
6
7  action JOIN_ROUND(n : node, r : round) {
8    assume r ≠ ⊥
9    assume start_round_msg(r)
10   assume ¬left_round(n, r) # rewritten
11   local maxr, v := max {(r', v') |
12                         vote_msg(n, r', v') ∧ r' < r}
13   join_ack_msg(n, r, maxr, v) := true
14   # generated update code for derived relations:
15   left_round(n, R) := left_round(n, R) ∨ R < r
16   joined_round(n, r) := true
17 }
18 action PROPOSE(r : round, q : quorum) {
19   assume r ≠ ⊥
20   assume ∀v. ¬propose_msg(r, v)
21   # rewritten to aviod quantifier alternation
22   assume ∀n. member(n, q) → joined_round(n, r)
23   # rewritten to use vote_msg instead of join_ack_msg
24   local maxr, v := max {(r', v') | ∃n. member(n, q)
25                         ∧vote_msg(n, r', v') ∧ r' < r}
26   propose_msg(r, v) := true
27 }
28 action VOTE(n : node, r : round, v : value) {
29   assume r ≠ ⊥
30   assume propose_msg(r, v)
31   assume ¬left_round(n, r) # rewritten
32   vote_msg(n, r, v) := true
33 }
```

Fig. 5. Changes to the Paxos model that allow verification in EPR. Declarations that appear in Fig. 3 are omitted, as well as the LEARN action which is left unmodified.

the use of sets of nodes as quantification over sets is also beyond first-order logic. We model these aspects according to the principles of Section 3.1:

Sorts and Axioms. We use the following four uninterpreted sorts: (i) node - to represent nodes of the system, (ii) value - to represent the values subject to the consensus algorithm, (iii) round - to model the rounds of Paxos, and (iv) quorum - to model sets of nodes with pairwise intersection in a first-order abstraction. While nodes and values are naturally uninterpreted, the rounds and the quorums are uninterpreted representations of interpreted concepts: integers and sets of nodes that intersect pairwise, respectively. We express some features that come from the desired interpretation using relations and axioms.

For rounds, we include a binary relation $\leq$, and axiomatize it to be a total order (Fig. 1). Our model also includes a constant $\bot$ of sort round, which represents a special round that is not considered an actual round of the protocol, and instead serves as a special value used in the join-acknowledgment (1b) message when a node has not yet voted for any value. Accordingly, any action assumes that the round it involves is not $\bot$.

The quorum sort is used to represent sets of nodes that contain strictly more than half of the nodes. As explained in Section 3.1, we introduce a membership relation between nodes and quorums. An important property for Paxos is that any two quorums intersect. We capture this with an axiom in first-order logic (Fig. 3 line 8).

State. The state of the protocol consists of the set of messages the nodes have sent. We represent these using relations, where each tuple in a relation corresponds to a single message. The relations *start_round_msg*, *join_ack_msg*, *propose_msg*, *vote_msg* correspond to the 1a, 1b, 2a, 2b phases of the algorithm, respectively. In modeling the algorithm, we assume all messages are sent to all nodes, so the relations do not contain destination fields. Note that recording messages via relations (i.e., sets) is an abstraction of the network but it is consistent with the messaging model we assume, in which messages may be lost, duplicated, and reordered. The *decision* relation captures the decisions learned by the nodes.

Actions. The different atomic steps taken by the nodes in the protocol are modeled using actions. The START_ROUND action models phase 1a of the protocol, sending a start round message to all nodes. The JOIN_ROUND action models the receipt of a start round message and the transmission of a join-acknowledgment (1b) message. The PROPOSE action models the receipt of join-acknowledgment (1b) messages from a quorum of nodes, and the transmission of a propose (2a) message which proposes a value for a round. The VOTE action models the receipt of a propose (2a) message by a node, and voting for a value by sending a vote (2b) message. Finally, the LEARN action models learning a decision by node n, when it is voted for by a quorum of nodes.

In these actions, sending a message is expressed by inserting the corresponding tuple to the corresponding relation. Different conditions (e.g., not joining a round if already joined higher round, properly reporting the previous votes, or appropriately selecting the proposed value) are expressed using assume statements. To prepare a join-acknowledgment message in JOIN_ROUND, as well as to propose a value in PROPOSE, a node needs to compute the maximal vote (performed by it or reported to it, respectively). This is done by a max operation (line 32 and line 44) which operates with respect to the order on rounds, and returns the round $\bot$ and an arbitrary value in case the set is empty. The $r, v := \max\{(r', v') \mid \varphi(r', v')\}$ operation is syntactic sugar for an assume of the following formula:

$$(r = \bot \wedge \forall r', v'.\ \neg\varphi(r', v')) \vee (r \neq \bot \wedge \varphi(r, v) \wedge \forall r', v'.\ \varphi(r', v') \rightarrow r' \leq r). \tag{3}$$

Note that if φ is a purely existentially quantified formula, then eq. (3) is alternation-free.

5.2 Inductive Invariant

The key safety property we wish to verify about Paxos is that only a single value can be decided (it can be decided at multiple rounds, as long as it is the same value). This is expressed by the following universally quantified formula:

$$\forall n_1, n_2 : \text{node}, r_1, r_2 : \text{round}, v_1, v_2 : \text{value}.\ \mathit{decision}(n_1, r_1, v_1) \wedge \mathit{decision}(n_2, r_2, v_2) \rightarrow v_1 = v_2 \tag{4}$$

While the safety property holds in all the reachable states of the protocol, it is not inductive. That is, assuming that it holds is not sufficient to prove that it still holds after an action is taken. For example, consider a state s in which $\mathit{decision}(n_1, r_1, v_1)$ holds and there is a quorum q of nodes such that, for every node n in q, $\mathit{vote\_msg}(n, r_2, v_2)$ holds, with $v_2 \neq v_1$. Note that the safety property holds in s. However, a LEARN action introduces a transition from state s to a state s' in which both $\mathit{decision}(n_2, r_1, v_1)$ and $\mathit{decision}(n_2, r_2, v_2)$ hold, violating the safety property. This counterexample to induction does not indicate a violation of safety, but it indicates that the safety property needs to be strengthened in order to obtain an inductive invariant. We now describe such an inductive invariant.

Our inductive invariant contains, in addition to the safety property, the following rather simple statements that are maintained by the protocol and are required for inductiveness:

$$\forall r : \text{round}, v_1, v_2 : \text{value}.\ \mathit{propose\_msg}(r, v_1) \wedge \mathit{propose\_msg}(r, v_2) \rightarrow v_1 = v_2 \tag{5}$$

$$\forall n : \text{node}, r : \text{round}, v : \text{value}.\ \mathit{vote\_msg}(n, r, v) \rightarrow \mathit{propose\_msg}(r, v) \tag{6}$$

$$\begin{aligned}&\forall r : \text{round}, v : \text{value}.\\ &\quad (\exists n : \text{node}.\ \mathit{decision}(n, r, v)) \rightarrow \exists q : \text{quorum}. \forall n : \text{node}.\ \mathit{member}(n, q) \rightarrow \mathit{vote\_msg}(n, r, v)\end{aligned} \tag{7}$$

Equation (5) states that there is a unique proposal per round. Equation (6) states that a vote for v in round r is cast only when a proposal for v has been made in round r. Equation (7) states that a decision for v is made in round r only if a quorum of nodes have voted for v in round r. In addition, the inductive invariant restricts the join-acknowledgment messages so that they faithfully represent the maximal vote (up to the joined round), or $\perp$ if there are no votes so far, and also asserts that there are no actual votes at round $\perp$:

$$\forall n : \text{node}, r, r' : \text{round}, v, v' : \text{value}.\ \mathit{join\_ack\_msg}(n, r, \perp, v) \wedge r' < r \rightarrow \neg \mathit{vote\_msg}(n, r', v') \tag{8}$$

$$\forall n : \text{node}, r, r' : \text{round}, v : \text{value}.\ \mathit{join\_ack\_msg}(n, r, r', v) \wedge r' \neq \perp \rightarrow r' < r \wedge \mathit{vote\_msg}(n, r', v) \tag{9}$$

$$\begin{aligned}&\forall n : \text{node}, r, r', r'' : \text{round}, v, v' : \text{value}.\\ &\quad \mathit{join\_ack\_msg}(n, r, r', v) \wedge r' \neq \perp \wedge r' < r'' < r \rightarrow \neg \mathit{vote\_msg}(n, r'', v')\end{aligned} \tag{10}$$

$$\forall n : \text{node}, v : \text{value}.\ \neg \mathit{vote\_msg}(n, \perp, v) \tag{11}$$

The properties stated so far are rather straightforward, and are usually not even mentioned in paper proofs or explanations of the protocol. The key to the correctness argument of the protocol is the observation that when the owner of round r_2 proposes a value in r_2, it cannot miss any value that is choosable at a lower round: whenever a value v_2 is proposed at round r_2, then in all rounds r_1 prior to r_2, no other value $v_1 \neq v_2$ is choosable. The property that no $v_1 \neq v_2$ is choosable at r_1 is captured in the inductive invariant by the requirement that in any quorum of nodes, there must be at least one node that has already left round r_1 (i.e., joined a higher round), and did not vote for v_1 at r_1 (and hence will also not vote for it in the future). Formally, this is:

$$\begin{aligned}&\forall r_1, r_2 : \text{round}, v_1, v_2 : \text{value}, q : \text{quorum}.\ \mathit{propose\_msg}(r_2, v_2) \wedge r_1 < r_2 \wedge v_1 \neq v_2 \rightarrow\\ &\exists n : \text{node}, r', r'' : \text{round}, v : \text{value}.\ \mathit{member}(n, q) \wedge \neg \mathit{vote\_msg}(n, r_1, v_1) \wedge r' > r_1 \wedge \mathit{join\_ack\_msg}(n, r', r'', v)\end{aligned} \tag{12}$$

The fact that this property is maintained by the protocol is obtained by the proposal mechanism and the interaction between phase 1 and phase 2 (see [Padon et al. 2017] for a detailed explanation).

Equations (4) to (12) define an inductive invariant that proves the safety of the Paxos model of Fig. 3. However, the verification condition for this inductive invariant contains cyclic quantifier alternations, and is therefore outside of EPR. We now review the quantifier alternations in the verification condition, which originate both from the model and from the inductive invariant.

In the model, the axiomatization of quorums (Fig. 3 line 8) introduces a $\forall\exists$-edge from quorum to node. In addition, the assumption in the PROPOSE action that join-acknowledgment messages were received from a quorum of nodes (line 39) introduces $\forall\exists$-edges from node to round and from node to value.

In the inductive invariant, only eqs. (7) and (12) include quantifier alternations (the rest are universally quantified). Equation (7) has quantifier structure $\forall$round, value $\exists$quorum $\forall$node[4]. Note that the inductive invariant appears both positively and negatively in the verification condition, so eq. (7) adds $\forall\exists$-edges from round to quorum and from value to quorum (from the positive occurrence), as well as an edge from quorum to node (from the negative occurrence). While the latter coincides with the edge that comes from the quorum axiomatization (line 8), the former edges closes a cycle in the quantifier alternation graph. Equation (12) has quantifier prefix $\forall$round, value, quorum $\exists$node, round, value. Thus, it introduces 9 edges in the quantifier alternation graph, including self-loops at round and value. In conclusion, while the presented model in first-order logic has an inductive invariant in first-order logic, the resulting verification condition is outside of EPR.

6 PAXOS IN EPR

The quantifier alternation graph of the model of Paxos described in Section 5 contains cycles. To obtain a safety proof of Paxos in EPR, we apply the methodology described in Section 3 to transform this model in a way that eliminates the cycles from the quantifier alternation graph. The resulting changes to the model are presented in Fig. 5, and the rest of this section explains them step by step.

6.1 Derived Relation for Left Rounds

We start by addressing the quantifier alternation that appears in eq. (12) as part of the inductive invariant. We observe that the following existentially quantified formula appears both as a subformula there, and in the conditions of the JOIN_ROUND and the VOTE actions (Fig. 3 lines 29 and 50):

$$\psi_1(n,r) = \exists r', r'' : \text{round}, v : \text{value}.\ r' > r \wedge \mathit{join\_ack\_msg}(n,r',r'',v)$$

This formula captures the fact that node n has joined a round higher than r, which means it promises to never participate in round r in any way, i.e., it will neither join nor vote in round r. We add a derived relation called *left_round* to capture ψ_1, so that $\mathit{left\_round}(n,r)$ captures the fact that node n has left round r. The formula ψ_1 is in the class of formulas handled by the scheme described in Section 3.3, and thus we obtain the initial condition and update code for *left_round*. The result appears in Fig. 5 lines 4 and 15.

Rewriting (steps 3+4). Using the *left_round* relation, we rewrite the conditions of the JOIN_ROUND and VOTE actions (Fig. 5 lines 10 and 31). These rewrites are trivially sound as explained in Section 3.2 (with a trivial rewrite condition). We also rewrite eq. (12) as:

$$\begin{aligned}&\forall r_1, r_2 : \text{round}, v_1, v_2 : \text{value}, q : \text{quorum}.\ \mathit{propose\_msg}(r_2, v_2) \wedge r_1 < r_2 \wedge v_1 \neq v_2 \rightarrow \\ &\quad \exists n : \text{node},\ \mathit{member}(n,q) \wedge \neg \mathit{vote\_msg}(n, r_1, v_1) \wedge \mathit{left\_round}(n, r_1)\end{aligned} \tag{13}$$

Equation (13) contains less quantifer alternations than eq. (12), and it will be part of the inductive invariant that will eventually be used to prove safety.

[4]The local existential quantifier in ($\exists n$: node.*decision*(n, r, v)) does not affect the quantifier alternation graph.

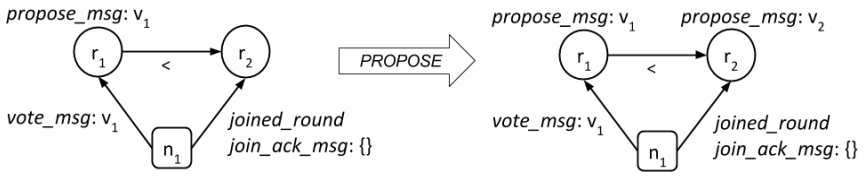

Fig. 6. Counterexample to induction of EPR model of Paxos after the first attempt. The counterexample contains one node n_1 (depicted as square), two rounds $r_1 < r_2$ (depicted as circles), two values v_1, v_2, and a single quorum (that contains n_1). The figure displays the the *join_ack_msg*, *joined_round*, *propose_msg*, *vote_msg* relations, as well as the $<$ relation (derived from $\leq$). The action occurring in the counterexample is PROPOSE, where an arbitrary value is proposed for r_2, even though node n_1 voted for v_1 in r_1. This erroneous behavior occurs due to the fact that the representation invariant of *joined_round* is violated in the pre-state of the counterexample: *joined_round*(n_1, r_2) holds, eventhough there is no corresponding *join_ack_msg* entry. This allows a PROPOSE action to erroneously propose v_2 for r_2, in spite of the fact that v_1 is choosable at r_1.

6.2 Derived Relation for Joined Rounds

After the previous transformation, the verification condition is still not stratified. The reason is the combination of eq. (13) and the condition of the PROPOSE action (Fig. 3 line 39):

$$\forall n : \text{node}.\ \mathit{member}(n, q) \rightarrow \exists r' : \text{round}, v : \text{value}.\ \mathit{join\_ack\_msg}(n, r, r', v).$$

While each of these introduces quantifier alternations that are stratified when viewed separately, together they form cycles. Equation (13) introduces $\forall\exists$-edges from round and value to node, while the PROPOSE condition introduces edges from node to round and value. The PROPOSE condition expresses the fact that every node in the quorum q has joined round r by sending a join-acknowledgment (1b) message to round r. However, because the join-acknowledgment message contains two more fields (representing the node's maximal vote so far), the condition existentially quantifies over them. To eliminate this existential quantification and remove the cycles, we add a derived relation, called *joined_round*, that captures the projection of *join_ack_msg* over its first two components, given by the formula:

$$\psi_2(n, r) = \exists r' : \text{round}, v : \text{value}.\ \mathit{join\_ack\_msg}(n, r, r', v)$$

This binary relation over nodes and rounds records the sending of join-acknowledgment messages, ignoring the maximal vote so far reported in the message. Thus, *joined_round*(n, r) captures the fact that node n has agreed to join round r. The formula ψ_2 is in the class of formulas handled by the scheme of Section 3.3, and thus we obtain the initial condition and update code for *joined_round*, as it appears in Fig. 5 lines 5 and 16.

Rewriting (steps 3+4): first attempt. We rewrite the condition of the PROPOSE action to use *joined_round* instead of ψ_2. (The rewrite condition is again trivial, ensuring soundness.) The result appears in Fig. 5 line 22, and is purely universally quantified. When considering the transformed model, and the candidate inductive invariant given by the conjunction of eqs. (4) to (11) and (13), the resulting quantifier alternation graph is acyclic. This means that the verification condition is in EPR and hence decidable to check. However, it turns out that this candidate invariant is not inductive, and the check yields a counterexample to induction.

The counterexample is depicted in Fig. 6. The counterexample shows a PROPOSE action that leads to a violation of eq. (13). The example contains a single node n_1 (which forms a quorum), that has voted for value v_1 in round r_1, and yet a different value v_2 is proposed for a later round r_2 (based on the quorum composed only of n_1) which leads to a violation of eq. (13). The PROPOSE action is enabled since *joined_round*(n_1, r_2) holds. However, an arbitrary value is proposed since

join_ack_msg is empty. The root cause for the counterexample is that the inductive invariant does not capture the connection between *joined_round* and *join_ack_msg*, so it allows a state in which a node n_1 has joined round r_2 according to *joined_round* (i.e., *joined_round*(n_1, r_2) holds), but it has not joined it according to *join_ack_msg* (i.e., $\exists r', v.$ *join_ack_msg*(n_1, r_2, r', v) does not hold). Note that the counterexample is spurious, in the sense that it does not represent a reachable state. However, for a proof by an inductive invariant, we must eliminate this counterexample nonetheless.

Rewriting (steps 3+4): second attempt. One obvious way to eliminate the counterexample discussed above is to add the representation invariant of *joined_round* to the inductive invariant. However, this will result in a cyclic quantifier alternation, causing the verification condition to be outside of EPR. Instead, we will eliminate this counterexample by rewriting the code of the PROPOSE action, relying on an auxiliary invariant to verify the rewrite, as explained in Section 3.2. We observe that the mismatch between *joined_round* and *join_ack_msg* is only problematic in this example because node n_1 voted in $r_1 < r_2$. While the condition of the PROPOSE action is supposed to ensure that the max operation considers past votes of all nodes in the quorum, such a scenario where the *joined_round* is inconsistent with *join_ack_msg* makes it possible for the PROPOSE action to overlook past votes, which is the case in this counterexample. Our remedy is therefore to rewrite the max operation (which is implemented by an assume command, as explained before) to consider the votes directly by referring to the *vote* messages instead of the *join-acknowledgment* messages that report them. We first formally state the rewrite and then justify its correctness.

As before, we rewrite the condition of the PROPOSE action to use *joined_round* (Fig. 5 line 22). In addition, we rewrite the max operation in Fig. 3 line 44, i.e., max $\{(r', v') \mid \varphi_1(r', v')\}$ where

$$\varphi_1(r', v') = \exists n.\ member(n, q) \wedge join\_ack\_msg(n, r, r', v') \wedge r' \neq \bot$$

to the max operation in Fig. 5 line 25, i.e., max $\{(r', v') \mid \varphi_2(r', v')\}$ where

$$\varphi_2(r', v') = \exists n.\ member(n, q) \wedge vote\_msg(n, r', v') \wedge r' < r$$

The key to the correctness of this change is that a join-acknowledgment message from node n to round r contains its maximal vote prior to round r, and once the node sent this message, it will never vote in rounds smaller than r. Therefore, while the original PROPOSE action considers the maximum over votes reflected by join-acknowledgment messages from a quorum, looking at the actual *votes* from the quorum in rounds prior to r (as captured by the *vote_msg* relation) yields the same maximum.

Formally, we establish the rewrite condition of step 3 given by eq. (2) using an auxiliary invariant INV_{aux}, defined as the conjunction of eqs. (5), (6) and (8) to (11). This invariant captures the connection between *join_ack_msg* and *vote_msg* explained above. The invariant INV_{aux} is inductive for the original model, and its verification condition is in EPR (the resulting quantifier alternation graph is acyclic). Second, we prove that under the assumption of INV_{aux} and the condition $\forall n.\ member(n, q) \rightarrow \exists r', v.join\_ack\_msg(n, r, r', v)$ (Fig. 3 line 39), the operation max$\{(r', v') \mid \varphi_1(r', v')\}$ is equivalent to the operation max$\{(r', v') \mid \varphi_2(r', v')\}$ (recall that both translate to assume's according to eq. (3)). This check is also in EPR. In conclusion, we are able to establish the rewrite condition using two EPR checks: one for proving INV_{aux}, and one for proving eq. (2).

Inductive Invariant. After the above rewrite, the conjunction of eqs. (4) to (11) and (13) is still not an inductive invariant, due to a counterexample to induction in which a node has joined a higher round according to *joined_round*, but has not left a lower round according to *left_round*. As before, the counterexample is inconsistent with the representation invariants. However, this time the counterexample (and another similar one) can be eliminated by strengthening the inductive invariant

with the following facts, which are implied by the representation invariants of *joined_round* and *left_round*:

$$\forall n : \text{node}, r_1, r_2 : \text{round}.\ r_1 < r_2 \wedge \mathit{joined\_round}(n, r_2) \rightarrow \mathit{left\_round}(n, r_1) \tag{14}$$

$$\forall n : \text{node}, r, r' : \text{round}, v : \text{value}.\ \mathit{join\_ack\_msg}(n, r, r', v) \rightarrow \mathit{joined\_round}(n, r) \tag{15}$$

Both are purely universally quantified and therefore do not affect the quantifier alternation graph.

Finally, the invariant given by the conjunction of eqs. (4) to (11) and (13) to (15) is indeed an inductive invariant for the transformed model. Fig. 4 depicts the quantifier alternation graph of the resulting verification condition. This graph is acyclic, and so the invariant can be verified in EPR. This invariant proves the safety of the transformed model (Fig. 5). Using Theorem 3.1, the safety of the original Paxos model (Fig. 3) follows.

7 MULTI-PAXOS

In this section we describe our verification of Multi-Paxos. Multi-Paxos is an implementation of state-machine replication (SMR): nodes run a sequence of instances of the Paxos algorithm, where the i^{th} instance is used to decide on the i^{th} command in the sequence of commands executed by the machine. For efficiency, when starting a round, a node uses a single message to simultaneously do so in all instances. In response, each node sends a join-acknowledgment message that reports its maximal vote in each instance; this message has finite size as there are only finitely many instances in which the node ever voted. The key advantage over using one isolated incarnation of Paxos per instance is that when a unique node takes on the responsibility of starting a round and proposing commands, only phase 2 of Paxos has to be executed for every proposal.

Below we provide a description of the EPR verification of Multi-Paxos. The main change compared to the Paxos consensus algorithm is in the modeling of the algorithm in first-order logic, where we use the technique of Section 3.1.2 to model higher-order concepts. The transformation to EPR is essentially the same as in Section 6, using the same derived relations and rewrites.

7.1 Model of the Protocol

Multi-Paxos uses the same message types as the basic Paxos algorithm, but when a node joins a round, it sends a join-acknowledgment message (1a) with its maximal vote for each instances (there will only be a finite set of instances for which it actually voted). Upon receipt of join-acknowledgment messages from a quorum of nodes that join round r, the owner of the round r determines the instances for which it is obliged to propose a value (because it may be choosable in a prior round), and the set of *available* instances for which it can propose any value. Next, the owner of round r can propose commands in available instances. This means that the PROPOSE action of Paxos is split into two actions in Multi-Paxos: one that processes the join-acknowledgment messages from a quorum and another that proposes new values.

Our model of Multi-Paxos in first-order logic appears in Fig. 7. We explain the key differences compared to the Paxos model from Fig. 3.

State. We extend the vocabulary of the Paxos model with two new sorts: instance and votemap. The instance sort represents instances, and the *propose_msg*, *vote_msg* and *decision* relations are extended to include an instance in each tuple. In practice, instances may be natural numbers that give the order in which commands of the state machine must be executed by each replica. However, we are only interested in proving consistency (i.e., that decisions are unique per instance), and the consistency proof does not depend on the instances being ordered. Therefore, our model does not include a total order over instances.

```
1  sort node
2  sort quorum
3  sort round
4  sort value
5  sort instance
6  sort votemap
7
8  relation ≤ : round, round
9  axiom total_order(≤)
10 constant ⊥ : round
11 relation member : node, quorum
12 axiom ∀q1, q2 : quorum. ∃n : node. member(n, q1) ∧ member(n, q2)
13
14 relation start_round_msg : round
15 relation join_ack_msg : node, round, votemap
16 relation propose_msg : instance, round, value
17 relation available : round, instance
18 relation vote_msg : node, instance, round, value
19 relation decision : node, instance, round, value
20 function roundof : votemap, instance → round
21 function valueof : votemap, instance → value
22
23 init ∀r. ¬start_round_msg(r)
24 init ∀n, r, m. ¬join_ack_msg(n, r, m)
25 init ∀i, r, v. ¬propose_msg(i, r, v)
26 init ∀r, i. ¬available(r, i)
27 init ∀n, i, r, v. ¬vote_msg(n, i, r, v)
28 init ∀r, v. ¬decision(r, v)
29
30 action START_ROUND(r : round) { assume r ≠ ⊥ ; start_round_msg(r) := true }
31 action JOIN_ROUND(n : node, r : round) {
32   assume r ≠ ⊥ ∧ start_round_msg(r) ∧ ¬∃r', m. r' > r ∧ join_ack_msg(n, r', m)
33   local m : votemap := *
34   assume ∀i. (roundof(m, i), valueof(m, i)) = max {(r', v') | vote_msg(n, i, r', v') ∧ r' < r}
35   join_ack_msg(n, r, m) := true
36 }
37 action INSTATE_ROUND(r : round, q : quorum) {
38   assume r ≠ ⊥
39   assume (∀i. ¬available(r, i)) ∧ (∀i, v. ¬propose_msg(i, r, v))
40   assume ∀n. member(n, q) → ∃m. join_ack_msg(n, r, m)
41   local m : votemap := *
42   assume ∀i. (roundof(m, i), valueof(m, i)) = max {(r', v') | ∃n, m'. member(n, q) ∧ join_ack_msg(n, r, m') ∧
43                                                              r' = roundof(m', i) ∧ v' = valueof(m', i) ∧ r' ≠ ⊥}
44   available(r, I) := (roundof(m, I) = ⊥)
45   propose_msg(I, r, V) := (roundof(m, I) ≠ ⊥ ∧ V = valueof(m, I))
46 }
47 action PROPOSE_NEW_VALUE(r : round, i : instance, v : value) {
48   assume r ≠ ⊥
49   assume available(r, i) ∧ ∀v. ¬propose_msg(i, r, v)
50   propose_msg(r, v) := true
51 }
52 action VOTE(n : node, i : instance, r : round, v : value) {
53   assume r ≠ ⊥ ∧ propose_msg(i, r, v) ∧ ¬∃r', m. r' > r ∧ join_ack_msg(n, r', m)
54   vote_msg(n, i, r, v) := true
55 }
56 action LEARN(n : node, i : instance, r : round, v : value, q : quorum) {
57   assume r ≠ ⊥ ∧ ∀n'. member(n', q) → vote_msg(n', i, r, v)
58   decision(n, i, r, v) := true
59 }
```

Fig. 7. Model of Multi-Paxos as a transition system in many-sorted first-order logic.

The votemap sort models a map from instances to (round, value) pairs, which are passed in the join-acknowledgment messages (captured by the relation *join_ack_msg* : node, round, votemap). We use the encoding explained in Section 3.1.2, and add two functions, *roundof* : votemap, instance → round and *valueof* : votemap, instance → value, that allow access to the content of a votemap. We also add a new relation *available* : round, instance, which records the instances each round owner considers to be available.

Actions. The START_ROUND action is identical to Paxos, and the VOTE and LEARN actions are identical except they are now parameterized by an instance. The JOIN_ROUND action is identical in principle, except it now must obtain and deliver a votemap that maps each instance to the maximal vote of the node (and $\perp$ for instances in which the node did not vote). To express the computation of the maximal vote of every instance i, we use an assume statement in line 34 of the form $\forall i.\ (roundof(m,i), valueof(m,i)) = \max\{(r',v') \mid \varphi(i,r',v')\}$ which follows a non-deterministic choice of m : votemap. The assume statement is realized by the following formula in the transition relation (which is an adaptation of eq. (3) to account for multiple instances):

$$\begin{aligned}&\forall i.\ (roundof(m,i) = \perp \wedge \forall r', v'.\ \neg\varphi(i,r',v')) \vee \\ &\quad (roundof(m,i) \neq \perp \wedge \varphi(i, roundof(m,i), valueof(m,i)) \wedge \forall r', v'.\ \varphi(i,r',v') \rightarrow r' \leq roundof(m,i))\end{aligned} \tag{16}$$

Note that for line 34, φ is quantifier free, so eq. (16) is purely universally quantified.

The most notable difference in the actions is that, in Multi-Paxos, the PROPOSE action is split into two actions: INSTATE_ROUND and PROPOSE_NEW_VALUE. INSTATE_ROUND processes the join-acknowledgment messages from a quorum, and PROPOSE_NEW_VALUE proposes new values, modeling the fact that only phase 2 is repeated for every instance.

The INSTATE_ROUND action takes place when the owner of a round received join-acknowledgment messages from a quorum of nodes. The owner then finds the maximal vote reported in the messages for each instance, which is done in line 42. This is realized using eq. (16). Observe that φ here contains existential quantifiers over node and votemap. This introduces quantifier alternation, which results in $\forall\exists$ edges from instance to both node and votemap. Fortunately, these edges do not create cycles in the quantifier alternation graph. Next, in line 44, all rounds for which no votes were reported are marked as available, and in line 45, all obligatory proposals are made.

The PROPOSE_NEW_VALUE action models the proposal of a new value of in an instance considered to be available by the round owner. This action occurs due to client requests, which are abstracted in our model.Therefore, the only preconditions of this action in our model are that the instance is considered available by the round owner (captured by the *available* relation), and that it has not proposed any other value for this instance. In practice, a round owner will choose the next available instance (according to some total order). However, since this is not necessarily for correctness, our model completely abstracts the total order over instances, and allows a new value to be proposed in any available instance.

7.2 Inductive Invariant

The safety property we wish to prove for Multi-Paxos is that each Paxos instance is safe. Formally:

$$\begin{aligned}&\forall i : \text{instance}, n_1, n_2 : \text{node}, r_1, r_2 : \text{round}, v_1, v_2 : \text{value}. \\ &\qquad decision(n_1, i, r_1, v_1) \wedge decision(n_2, i, r_2, v_2) \rightarrow v_1 = v_2\end{aligned} \tag{17}$$

Equation (17) generalizes eq. (4) by universally quantifying over all instances. The inductive invariant that proves safety contains similarly generalized versions of eqs. (5) to (10) and (12), where the *join_ack_msg* relation is adjusted to contain a votemap element instead of a round, value

pair as the message content. In addition, for inductiveness, we must capture the connection between the *available* relation and past votes. Namely, that whenever an instance i is marked available in some round r, then no value is choosable for instance i at any round lower than r. Formally:

$$\begin{aligned}&\forall i : \text{instance}, r_1, r_2 : \text{round}, v : \text{value}, q : \text{quorum}.\ \mathit{available}(r_2, i) \wedge r_1 < r_2 \rightarrow \\ &\quad \exists n : \text{node}, r' : \text{round}, m : \text{votemap}.\ \mathit{member}(n, q) \wedge \neg \mathit{vote\_msg}(n, r_1, v) \wedge r' > r_1 \wedge \\ &\qquad\qquad \mathit{join\_ack\_msg}(n, r', m)\end{aligned} \tag{18}$$

7.3 Transformation to EPR

As with the Paxos model of Fig. 3, the resulting verification condition for the Multi-Paxos model is outside of EPR, and must be transformed to allow EPR verification. The required transformations are essentially identical to the Paxos model (the new sorts do not appear in any cycles in the quantifier alternation graph), where we define the *left_round* relation by:

$$\psi_1(n, r) = \exists r', m.\ r' > r \wedge \mathit{join\_ack\_msg}(n, r', m)$$

And the *joined_round* relation by:

$$\psi_2(n, r) = \exists m.\ \mathit{join\_ack\_msg}(n, r, m)$$

The last step required for EPR verification is to rewrite the max operation in line 42 of Fig. 7 to use *vote_msg* instead of *join_ack_msg*, which is exactly the same change that was required to verify the Paxos model. Thus, the transformations to EPR are in this case completely reusable, and allow EPR verification of Multi-Paxos.

8 PAXOS VARIANTS

In this section we briefly describe our verification in EPR of several variants of Paxos. More elaborate explanations are provided in the extended version of this paper [Padon et al. 2017]. In all cases, the transformations to EPR of Section 6 were employed (with slight modifications), demonstrating the reusability of the derived relations and rewrites across different Paxos variants.

8.1 Vertical Paxos

Vertical Paxos [Lamport et al. 2009] is a variant of Paxos whose set of participating nodes and quorums (called the configuration) can be changed dynamically by an external reconfiguration master. By using reconfiguration to replace failed nodes, Vertical Paxos makes Paxos reliable in the long-term. The reconfiguration master dynamically assigns configurations to rounds, which means that each round uses a different set of quorums. A significant algorithmic complication is that old configurations must be eventually retired in practice. This is achieved by having the nodes inform the master when a round r becomes *complete*, meaning that r holds all the necessary information about choosable values in lower rounds. The master tracks the highest complete round and passes it on to each new configuration to indicate that lower rounds need not be accessed. Rounds below the highest complete round can then be retired safely.

We model configurations in first-order logic by introducing a new sort config that represents a set of quorums, with a suitable member relation called *quorum_in*. Moreover, we change the axiomatization of quorums to only require that quorums of the same configuration intersect:

$$\begin{aligned}&\forall c : \text{config}, q_1, q_2 : \text{quorum}.\ \mathit{quorum\_in}(q_1, c) \wedge \mathit{quorum\_in}(q_2, c) \rightarrow \\ &\qquad \exists n : \text{node}.\, \mathit{member}(n, q_1) \wedge \mathit{member}(n, q_2)\end{aligned}$$

To model the complete round associated to each configuration, we introduce a function symbol $\mathit{complete\_of} : \text{config} \rightarrow \text{round}$. This function symbol introduces additional cycles to the quantifier

alternation graph. The transformation to EPR replaces the *complete_of* function by a derived relation defined by the formula $complete\_of(c) = r$. With this derived relation, we can rewrite the model and invariant so that the function no longer appears in the verification condition, and hence the cycles that it introduced are eliminated. Other than that, the transformation to EPR uses the same derived relations and rewrites of Section 6 (in fact, it only requires the *left_round* derived relation). A full description appears in [Padon et al. 2017].

8.2 Fast Paxos

Fast Paxos [Lamport 2006] is a variant of Multi-Paxos that improves its latency. The key idea is to mark some of the rounds as *fast* and allow any node to directly propose values in these rounds without going through the round owner. As a result, multiple values can be proposed, as well as voted for, in the same (fast) round. In order to maintain consistency, Fast Paxos uses two kinds of quorums: *classic* quorums and *fast* quorums. The quorums have the property that any two classic quorums intersect, and any classic quorum and *two* fast quorums intersect. Now, in a propose action receiving join-acknowledgment messages from the classic quorum q with a maximal vote reported in a fast round *maxr*, multiple different values may be reported by nodes in q in *maxr*. To determine which one may be choosable in *maxr*, a node will check whether there exists a *fast* quorum f such that all the nodes in $q \cap f$ reported voting v in *maxr*. If yes, by the intersection property of quorums, only this value v may be choosable in *maxr*, and hence must be proposed.

We model fast quorums with an additional sort f_quorum (and relation *f_member*), and axiomatize its intersection property as:

$$\forall q : \text{quorum}, f_1, f_2 : \text{f\_quorum}.\ \exists n : \text{node}.\ member(n, q) \wedge f\_member(n, f_1) \wedge f\_member(n, f_2)$$

The rest of the details of the model appear in [Padon et al. 2017]. The transformation to EPR is similar to Section 6. This includes the rewrite of the new condition for proposing a value. Interestingly, in this case, the verification of the latter rewrite is not in EPR when considering the formulas as a whole, but it is in EPR when we consider only the subformulas that change (see [Padon et al. 2017]).

8.3 Flexible Paxos

Flexible Paxos [Howard et al. 2016] extends Paxos based on the observation that it is only necessary that the quorums used in phase 1 intersect with the quorums used in phase 2 (as initially observed in [Lampson 2001]). This allows greater flexibility, and introduces a trade-off between the cost of deciding on new values and the cost of starting a new round. For example, in a system with 10 nodes, one may use sets of 8 nodes as phase 1 quorums, and sets of 3 nodes phase 2 quorums. EPR verification of Flexible Paxos is essentially the same as for normal Paxos, except we introduce two quorum sorts (for phase 1 and phase 2), and adapt the intersection axiom to:

$$\forall q_1 : \text{quorum}_1, q_2 : \text{quorum}_2.\ \exists n : \text{node}.\ member_1(n, q_1) \wedge member_2(n, q_2)$$

The detailed model and the adjusted invariant appear in [Padon et al. 2017].

8.4 Stoppable Paxos

Stoppable Paxos [Lamport et al. 2008] extends Multi-Paxos with the ability for a node to propose a special stop command in order to stop the algorithm, with the guarantee that if the stop command is decided in instance i, then no command is ever decided at an instance $j > i$. Stoppable Paxos therefore enables Virtually Synchronous system reconfiguration [Birman 2010; Chockler et al. 2001]: Stoppable Paxos stops in a state known to all participants, which can then start a new instance of Stoppable Paxos in a new configuration (e.g., in which participants have been added or removed);

	EPR		INV_{aux}		**RW**		**FOL** − 2			**FOL** − 4			**FOL** − 8			**FOL** − 16		
Protocol	μ	σ	μ	σ	μ	σ	μ	σ	τ	μ	σ	τ	μ	σ	τ	μ	σ	τ
Paxos	1.0	0.3	0.5	0	0.3	0	1.0	0.1	0	4.7	4.9	0	82	113	2	300	0	10
Multi-Paxos	1.2	0.1	0.6	0	0.7	0.2	1.4	0.2	0	6.0	1.4	0	230	109	7	300	0	10
Vertical Paxos	2.3	0.3	–	–	–	–	26	10	0	45	18	0	300	0	10	300	0	10
Fast Paxos	3.6	0.9	0.7	0	0.4	0	3.7	1.0	0	14	9.5	0	209	109	5	300	0	10
Flexible Paxos	0.8	0	0.5	0	0.3	0	0.8	0.1	0	1.7	0.9	0	65	98	1	229	113	7
Stoppable Paxos	5.1	3.7	0.9	0.1	0.5	0.1	163	134	4	300	0	10	300	0	10	300	0	10

Fig. 8. Run times (in seconds) of checking verification conditions using IVy and Z3. Each experiment was repeated 10 times (with random seeds used for Z3's heuristics). μ reports the mean time, σ reports the standard deviation, and τ reports the number of runs that timed out at 300 seconds (where this occurred). **EPR** is the verification of the EPR model. INV_{aux} is the verification of the auxiliary invariant. **RW** is the verification of the rewrite condition. **FOL** − N is the run time of semi-bounded verification of the first-order model, with bound 2 for values and bound N for rounds (in all variants, bounding the number of values and rounds eliminates cycles from the quantifier alternation graph).

moreover, no pending commands can leak from a configuration to the next, as only the final state of the command sequence is transfered from one configuration to the next.

Stoppable Paxos may be the most intricate algorithm in the Paxos family: as acknowledged by Lamport et al. [Lamport et al. 2008], "getting the details right was not easy". The main algorithmic difficulty in Stoppable Paxos is to ensure that no command may be decided after a stop command while at the same time allowing a node to propose new commands without waiting, when earlier commands are still in flight (which is important for performance). In contrast, in the traditional approach to reconfigurable SMR [Lamport et al. 2010], a node that has c outstanding command proposals may cause up to c commands to be decided after a stop command is decided; Those commands needs to be passed-on to the next configuration and may contain other stop commands, adding to the complexity of the reconfiguration system.

Before proposing a command in an instance in Stoppable Paxos, a node must check if other instances have seen stop commands proposed and in which round. This creates a non-trivial dependency between rounds and instances, which are mostly orthogonal concepts in other variants of Paxos. This manifest as the instance sort having no incoming edge in the quantifier alternation graph in other variants, while such edges appear in Stoppable Paxos. Interestingly, the rule given by Lamport et al. to propose commands results in verification conditions that are not in EPR, and rewriting seems difficult. However, we found an alternative rule which results in EPR verification conditions. This alternative rule soundly overapproximates the original rule (and introduces new behaviors), and, as we have verified (in EPR), it also maintains safety. The details of the modified rule and its verification appear in [Padon et al. 2017].

9 EXPERIMENTAL EVALUATION

We have implemented our methodology using the IVy tool [McMillan 2016; Padon et al. 2016], which uses the Z3 theorem prover [de Moura and Bjørner 2008] for checking verification conditions. Fig. 8 lists the run times for the automated checks performed when verifying the different Paxos variants. The experiments were performed on a laptop running Linux, with a Core-i7 1.8 GHz CPU. Z3 version 4.5.0 was used, along with the latest version of IVy. Z3 uses heuristics which employ randomness. Therefore, each experiment was repeated 10 times using random seeds. We report the mean times, as well as the standard deviation and the number of experiments which timed out at

300 seconds (these are included in the mean). The IVy files used for these experiments are available at the supplementary web page of this paper[5].

For each variant, Fig. 8 reports the time for checking the inductive invariant that proves the safety of the EPR model, as well as the times required to verify auxiliary invariants and rewrite conditions (see Section 3.2). We also report on the times required to check the inductive invariant for the original first-order logic models, using semi-bounded verification. In all variants, quantifier alternation cycles can be eliminated by bounding the number of values and rounds. We bound the number of values to 2, and vary the bound on the number of rounds.

As Fig. 8 demonstrates, using our methodology for EPR verification results in verification conditions that are solved by Z3 in a few seconds, with no timeouts, and with negligible variance among runs. In contrast, when using semi-bounded verification, the run time quickly increases as we attempt to increase the number of rounds. Moreover, the variance in run time increases significantly, causing an unpredictable experience for verification users. We have also attempted to use unbounded verification for the first-order logic models, but Z3 diverged in this case for all variants. This shows the practical value of our methodology, as it allows to transform models whose verification condition cannot be handled by Z3 (and demonstrate poor scalability and predictability for bounded verification), into models that can be verified by Z3 in a few seconds.

10 RELATED WORK

Automated verification of distributed protocols. Here we review several works that developed techniques for automated verification of distributed protocols, and compare them with our approach.

The Consensus Verification Logic $\mathbb{CL}$ [Dragoi et al. 2014] is a logic tailored to verify consensus algorithms in the partially synchronous Heard-Of Model [Charron-Bost and Schiper 2009], with a decidable fragment that can be used for verification. PSync [Dragoi et al. 2016] is a domain-specific programming language and runtime for developing formally verified implementations of consensus algorithms based on $\mathbb{CL}$ and the Heard-Of Model. Once the user provides inductive invariants and ranking functions in $\mathbb{CL}$, safety and liveness can be automatically verified. PSync's verified implementation of LastVoting (Paxos in the Heard-Of Model) is comparable in performance with state-of-the-art unverified systems.

Many interesting theoretical decidability results, as well as the ByMC verification tool, have been developed based on the formalism of Threshold Automata [Bloem et al. 2015; Konnov et al. 2017, 2015a,b]. This formalism allows to express a restricted class of distributed algorithms operating in a partially synchronous communication mode. This restriction allows decidability results based on cutoff theorems, for both safety and liveness.

[Alberti et al. 2016] present a decidable fragment of Presburger arithmetic with function symbols and cardinality constraint over interpreted sets. Their work is motivated by applications to the verification of fault-tolerant distributed algorithms, and they demonstrate automatic safety verification of some fault-tolerant distributed algorithms expressed in a partially synchronous round-by-round model similar to PSync.

#Π [v. Gleissenthall et al. 2016] present a logic that combines reasoning about set cardinalities and universal quantifiers, along with an invariant synthesis method. The logic is not decidable, so a sound and incomplete reasoning method is used to check inductive invariants. Inductive invariants are automatically synthesized by method of Horn constraint solving. The technique is applied to automatically verify a collection of parameterized systems, including mutual exclusion, consensus, and garbage collection. However, Paxos-like algorithms are beyond the reach of this verification methodology since they require more complicated inductive invariants.

[5] http://www.cs.tau.ac.il/~odedp/paxos-made-epr.html

Recently, [Maric et al. 2017] presented a cutoff result for consensus algorithms. They define *ConsL*, a domain specific language for consensus algorithms, whose semantics is based on the Heard-Of Model. *ConsL* admits a cutoff theorem, i.e., a parameterized algorithm expressed in *ConsL* is correct (for any number of processors) if and only if it is correct up to a some finite bounded number of processors (e.g., for Paxos the bound is 5) . This theoretical result shows that for algorithms expressible in *ConsL*, verification is decidable. However, *ConsL* is focused on algorithms for the core consensus problem, and does not support the infinite-state per process that is needed, e.g., to model Multi-Paxos and SMR.

The above mentioned works obtain automation (and some decidability) by restricting the programming model. We note that our approach takes a different path to decidability compared to these works. We axiomatize arithmetic, set cardinalities, and other higher-order concepts in an uninterpreted first-order abstraction. This is in contrast to the above works, in which these concepts are baked into specially designed logics and formalisms. Furthermore, we start with a Turing-complete modeling language and invariants with unrestricted quantifier alternation, and provide a methodology to reduce quantifier alternation to obtain decidability. This allows us to employ a general-purpose decidable logic to verify asynchronous Paxos, Multi-Paxos, and their variants, which are beyond the reach of all of the above works.

Deductive verification in undecidable logic. IronFleet [Hawblitzel et al. 2015] is a verified implementation of SMR, using the Dafny [Leino 2010] program verifier, with verified safety and liveness properties. Compared to our work, this system implementation is considerably more detailed. The verification using Dafny employs Z3 to check verification conditions expressed in undecidable logics that combine multiple theories and quantifier alternations. This leads to great difficulties due to the unpredictability of the solver. To mitigate some of this unpredictability, IronFleet adopted a style they call *invariant quantifier hiding*. This style attempts to specify the invariants in a way that reduces the quantifiers that are explicitly exposed to the solver. Our work is motivated by the IronFleet experience and observations. The methodology we develop provides a more systematic treatment of quantifier alternations, and reduces the verification conditions to a decidable logic.

Verification using interactive theorem provers. Recently, the Coq [Bertot and Castéran 2004] proof assistant has been used to develop verified implementations of systems, such as a file system [Chen et al. 2016], and shared memory data structures [Sergey et al. 2015]. Closer to our work is Verdi [Wilcox et al. 2015], which presents a verified implementation of an SMR system based on Raft [Ongaro and Ousterhout 2014], a Paxos-like algorithm. This approach requires great effort, due to the manual process of the proof; developing a verified SMR system requires many months of work by verification experts, and proofs measure in thousands of lines.

[Rahli et al. 2015; Schiper et al. 2014] verify the safety of implementations of consensus and SMR algorithms in the EventML programming language. EventML interfaces with the Nuprl theorem prover, in which proofs are conducted, and uses Nuprl's code generation facilities.

Other works applied interactive theorem proving to verify Paxos protocols at the algorithmic level, without an executable implementation. [Jaskelioff and Merz 2005] proved the correctness of the Disk Paxos algorithm in Isabelle/HOL [Nipkow et al. 2002], in about 6,500 lines of proof script. Recently, [Chand et al. 2016] presented safety proofs of Paxos and Multi-Paxos using the TLA+ [Lamport 1994] specification language and the TLA Proof System TLAPS [Chaudhuri et al. 2010]. TLA+ has also been used in Amazon to model check distributed algorithms [Newcombe et al. 2015]. However, they did not spend the effort required to obtain formal proofs, and only used the TLA+ models for bug finding via the TLA+ model checker [Yu et al. 1999].

Compared to our approach, using interactive theorem provers requires more user expertise and is more labor intensive. We note that part of the difficulty in using an interactive theorem prover lies

in the unpredictability of the automated proof methods available and the considerable experience needed to write proofs in an idiomatic style that facilitates automation. An interesting direction of research is to integrate our methodology in an interactive theorem prover to achieve predictable automation in a style that is natural to systems designers.

Works based on EPR. [Padon et al. 2016] and [McMillan 2016] have also used EPR to verify distributed protocols and cache coherence protocols. [Padon et al. 2016] develops an interactive technique for assisting the user to find universally quantified invariants (without quantifier alternations). In contrast, here we use invariants that contain quantifier alternations, as used in proofs of Paxos protocols. [McMillan 2016] goes beyond our work by extracting executable code from the modeling language. In the future, we plan to apply a similar extraction methodology to Paxos protocols.

In [Itzhaky 2014; Itzhaky et al. 2014, 2013] it was shown that EPR can express a limited form of transitive closure, in the context of linked lists manipulations. We notice that in the context of our methodology, their treatment of transitive closure can be considered as adding a derived relation. This work and our work both show that EPR is surprisingly powerful, when augmented with derived relations.

In [Feldman et al. 2017], bounded quantifier instantiation is explored as a possible solution to the undecidability caused by quantifier alternations. This work shares some of the motivation and challenges with our work, but proposes an alternative solution. The context we consider here is also wider, since we deal not only with quantifier alternations in the inductive invariant, but also with quantifier alternations in the transition relation. [Feldman et al. 2017] also shows an interesting connection between derived relations and quantifier instantiation, and these insights may apply to our methodology as well. An appealing future research direction is to combine user provided derived relations and rewrites together with heuristically generated quantifier instantiation.

11 CONCLUSION

In this paper we have shown how to verify interesting distributed protocols using EPR—a decidable fragment of first-order logic, which is supported by existing solvers (e.g., [Barrett et al. 2011; de Moura and Bjørner 2008; Korovin 2008; Riazanov and Voronkov 2002]). To mitigate the gap between the complexity of Paxos-like protocols and the restrictions of EPR, we developed a methodology for gradually eliminating complications. While this process requires assistance from the user, its steps are also mechanically checked (in EPR) to guarantee soundness.

We believe that our methodology can be applied to other distributed protocols as well, as our setting is very general. The generality of our approach is rooted in the use of first-order logic, with arbitrary relations and functions, and a Turing-complete imperative language.

While EPR has shown to be surprisingly powerful, we note that it is not a panacea, as some cyclic quantifier alternations may not be avoided. Still, we have successfully used our methodology to eliminate the cycles from several variants of Paxos, which is considered a rich and complex protocol of great practical importance.

ACKNOWLEDGMENTS

We thank Yotam M. Y. Feldman, Ken McMillan, Yuri Meshman, James R. Wilcox, and the anonymous referees for insightful comments which improved this paper. Padon and Sagiv were supported by the European Research Council under the European Union's Seventh Framework Program (FP7/2007–2013) / ERC grant agreement no. [321174-VSSC]. This research was partially supported by Len Blavatnik and the Blavatnik Family foundation. This material is based upon work supported by the National Science Foundation under Grant No. 1655166.

REFERENCES

Francesco Alberti, Silvio Ghilardi, and Elena Pagani. 2016. Counting Constraints in Flat Array Fragments. In *Automated Reasoning*. Springer, Cham, 65–81.

Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. 2011. CVC4. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*. 171–177.

Yves Bertot and Pierre Castéran. 2004. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Springer. https://doi.org/10.1007/978-3-662-07964-5

Ken Birman. 2010. A History of the Virtual Synchrony Replication Model. In *Replication: Theory and Practice (Lecture Notes in Computer Science)*, Bernadette Charron-Bost, Fernando Pedone, and André Schiper (Eds.), Vol. 5959. Springer, 91–120. https://doi.org/10.1007/978-3-642-11294-2_6

Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. 2015. *Decidability of Parameterized Verification*. Morgan & Claypool Publishers. https://doi.org/10.2200/S00658ED1V01Y201508DCT013

Michael Burrows. 2006. The Chubby Lock Service for Loosely-Coupled Distributed Systems. In *7th Symposium on Operating Systems Design and Implementation OSDI '06), November 6-8, Seattle, WA, USA*. USENIX Association, 335–350.

Saksham Chand, Yanhong A. Liu, and Scott D. Stoller. 2016. Formal Verification of Multi-Paxos for Distributed Consensus. In *FM 2016: Formal Methods: 21st International Symposium, Limassol, Cyprus, November 9-11, 2016, Proceedings 21*. Springer, 119–136.

Bernadette Charron-Bost and AndrÃ© Schiper. 2009. The Heard-of Model: Computing in Distributed Systems with Benign Faults. *Distributed Computing* 22, 1 (2009), 49–71.

Kaustuv Chaudhuri, Damien Doligez, Leslie Lamport, and Stephan Merz. 2010. The TLA+Proof System: Building a Heterogeneous Verification Platform. In *Proceedings of the 7th International Colloquium Conference on Theoretical Aspects of Computing (ICTAC'10)*. Springer-Verlag, 44–44.

Haogang Chen, Daniel Ziegler, Tej Chajed, Adam Chlipala, M. Frans Kaashoek, and Nickolai Zeldovich. 2016. Using Crash Hoare Logic for Certifying the FSCQ File System. In *2016 USENIX Annual Technical Conference, USENIX ATC 2016, Denver, CO, USA, June 22-24, 2016*.

Gregory V. Chockler, Idit Keidar, and Roman Vitenberg. 2001. Group communication specifications: a comprehensive study. *ACM Comput. Surv.* 33, 4 (2001), 427–469. https://doi.org/10.1145/503112.503113

Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings (Lecture Notes in Computer Science)*, Vol. 4963. Springer, 337–340.

Cezara Dragoi, Thomas A. Henzinger, Helmut Veith, Josef Widder, and Damien Zufferey. 2014. A Logic-Based Framework for Verifying Consensus Algorithms. In *International Conference on Verification, Model Checking, and Abstract Interpretation*. Springer, 161–181.

Cezara Dragoi, Thomas A. Henzinger, and Damien Zufferey. 2016. PSync: A Partially Synchronous Language for Fault-Tolerant Distributed Algorithms. *ACM SIGPLAN Notices* 51, 1 (2016), 400–415.

Yotam M. Y. Feldman, Oded Padon, Neil Immerman, Mooly Sagiv, and Sharon Shoham. 2017. Bounded Quantifier Instantiation for Checking Inductive Invariants. In *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I (Lecture Notes in Computer Science)*, Axel Legay and Tiziana Margaria (Eds.), Vol. 10205. 76–95. https://doi.org/10.1007/978-3-662-54577-5_5

Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch, Bryan Parno, Michael L. Roberts, Srinath T. V. Setty, and Brian Zill. 2015. IronFleet: proving practical distributed systems correct. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP*. 1–17.

Heidi Howard, Dahlia Malkhi, and Alexander Spiegelman. 2016. Flexible Paxos: Quorum Intersection Revisited. *arXiv preprint arXiv:1608.06696* (2016).

Shachar Itzhaky. 2014. *Automatic Reasoning for Pointer Programs Using Decidable Logics*. Ph.D. Dissertation. Tel Aviv University.

Shachar Itzhaky, Anindya Banerjee, Neil Immerman, Ori Lahav, Aleksandar Nanevski, and Mooly Sagiv. 2014. Modular reasoning about heap paths via effectively propositional formulas. In *the 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL*. 385–396.

Shachar Itzhaky, Anindya Banerjee, Neil Immerman, Aleksandar Nanevski, and Mooly Sagiv. 2013. Effectively-Propositional Reasoning about Reachability in Linked Data Structures. In *CAV (LNCS)*, Vol. 8044. 756–772.

Daniel Jackson. 2006. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press.

Mauro Jaskelioff and Stephan Merz. 2005. Proving the Correctness of Disk Paxos. *Archive of Formal Proofs* (June 2005). http://isa-afp.org/entries/DiskPaxos.shtml, Formal proof development.

Igor Konnov, Marijana Lazic, Helmut Veith, and Josef Widder. 2017. A Short Counterexample Property for Safety and Liveness Verification of Fault-Tolerant Distributed Algorithms. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017)*. ACM, 719–734.

Igor Konnov, Helmut Veith, and Josef Widder. 2015a. SMT and POR Beat Counter Abstraction: Parameterized Model Checking of Threshold-Based Distributed Algorithms. In *Computer Aided Verification*. Springer, Cham, 85–102.

Igor V. Konnov, Helmut Veith, and Josef Widder. 2015b. What You Always Wanted to Know About Model Checking of Fault-Tolerant Distributed Algorithms. In *Perspectives of System Informatics - 10th International Andrei Ershov Informatics Conference, PSI 2015, in Memory of Helmut Veith, Kazan and Innopolis, Russia, August 24-27, 2015, Revised Selected Papers (Lecture Notes in Computer Science)*, Manuel Mazzara and Andrei Voronkov (Eds.), Vol. 9609. Springer, 6–21. https://doi.org/10.1007/978-3-319-41579-6_2

Konstantin Korovin. 2008. iProver - An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*. 292–298.

Leslie Lamport. 1994. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 16, 3 (1994), 872–923.

Leslie Lamport. 1998. The Part-Time Parliament. *ACM Trans. Comput. Syst.* 16, 2 (1998), 133–169. https://doi.org/10.1145/279227.279229

Leslie Lamport. 2001. Paxos Made Simple. (December 2001), 51–58. https://www.microsoft.com/en-us/research/publication/paxos-made-simple/

Leslie Lamport. 2002. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Leslie Lamport. 2006. Fast Paxos. *Distributed Computing* 19, 2 (2006), 79–103.

Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. 2008. *Stoppable Paxos*. Technical Report. TechReport, Microsoft Research. https://www.microsoft.com/en-us/research/publication/stoppable-paxos/

Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. 2009. Vertical Paxos and Primary-Backup Replication. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing*. ACM, 312–313.

Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. 2010. Reconfiguring a State Machine. *SIGACT News* 41, 1 (03 2010), 63–73.

Butler Lampson. 2001. The ABCD's of Paxos. In *PODC*, Vol. 1. 13.

K Rustan M Leino. 2010. Dafny: An automatic program verifier for functional correctness. In *Logic for Programming, Artificial Intelligence, and Reasoning*. Springer, 348–370.

Harry R. Lewis. 1980. Complexity results for classes of quantificational formulas. *J. Comput. System Sci.* 21, 3 (1980), 317 – 353.

Ognjen Maric, Christoph Sprenger, and David A. Basin. 2017. Cutoff Bounds for Consensus Algorithms. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II (Lecture Notes in Computer Science)*, Rupak Majumdar and Viktor Kuncak (Eds.), Vol. 10427. Springer, 217–237. https://doi.org/10.1007/978-3-319-63390-9_12

Kenneth L. McMillan. 2016. Modular specification and verification of a cache-coherent interface. In *2016 Formal Methods in Computer-Aided Design, FMCAD 2016, Mountain View, CA, USA, October 3-6, 2016*, Ruzica Piskac and Muralidhar Talupur (Eds.). IEEE, 109–116. https://doi.org/10.1109/FMCAD.2016.7886668

Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. 2015. How Amazon web services uses formal methods. *Commun. ACM* 58, 4 (2015), 66–73.

Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. 2002. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Vol. 2283. Springer Science & Business Media.

Diego Ongaro and John K. Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014*. 305–319. https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

Oded Padon, Giuliano Losa, Mooly Sagiv, and Sharon Shoham. 2017. *Paxos made EPR: Decidable Reasoning about Distributed Protocols*. Technical Report.

Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. 2016. Ivy: safety verification by interactive generalization. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*. 614–630.

Robert Paige and Shaye Koenig. 1982. Finite Differencing of Computable Expressions. *ACM Trans. Program. Lang. Syst.* 4, 3 (1982), 402–454.

Ruzica Piskac, Leonardo Mendonça de Moura, and Nikolaj Bjørner. 2010. Deciding Effectively Propositional Logic Using DPLL and Substitution Sets. *J. Autom. Reasoning* 44, 4 (2010), 401–424.

Vincent Rahli, David Guaspari, Mark Bickford, and Robert L. Constable. 2015. 15th international workshop on automated verification of critical systems (avocs 2015). *electronic communications of the easst* 72 (2015).

Thomas W. Reps, Mooly Sagiv, and Alexey Loginov. 2010. Finite differencing of logical formulas for static analysis. *ACM Trans. Program. Lang. Syst.* 32, 6 (2010).

Alexandre Riazanov and Andrei Voronkov. 2002. The Design and Implementation of VAMPIRE. *AI Commun.* 15, 2,3 (Aug. 2002), 91–110. http://dl.acm.org/citation.cfm?id=1218615.1218620

N. Schiper, V. Rahli, R. V. Renesse, M. Bickford, and R. L. Constable. 2014. developing correctly replicated databases using formal tools. In *2014 44th annual ieee/ifip international conference on dependable systems and networks*. 395–406.

Fred B. Schneider. 1990. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys (CSUR)* 22, 4 (1990), 299–319.

Ilya Sergey, Aleksandar Nanevski, and Anindya Banerjee. 2015. Mechanized verification of fine-grained concurrent programs. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, David Grove and Steve Blackburn (Eds.). ACM, 77–87. https://doi.org/10.1145/2737924.2737964

Klaus v. Gleissenthall, NikolajBjørner Bjørner, and Andrey Rybalchenko. 2016. Cardinalities and Universal Quantifiers for Verifying Parameterized Systems. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '16)*. ACM, 599–613.

Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. 2009. SPASS Version 3.5. In *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*. 140–145.

James R. Wilcox, Doug Woos, Pavel Panchekha, Zachary Tatlock, Xi Wang, Michael D. Ernst, and Thomas E. Anderson. 2015. Verdi: a framework for implementing and formally verifying distributed systems. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*. 357–368.

Yuan Yu, Panagiotis Manolios, and Leslie Lamport. 1999. Model Checking TLA$^+$ Specifications. In *Correct Hardware Design and Verification Methods, 10th IFIP WG 10.5 Advanced Research Working Conference, CHARME '99, Bad Herrenalb, Germany, September 27-29, 1999, Proceedings (Lecture Notes in Computer Science)*, Laurence Pierre and Thomas Kropf (Eds.), Vol. 1703. Springer, 54–66. https://doi.org/10.1007/3-540-48153-2_6