Towards an Automatic Proof of Lamport’s Paxos

Aman Goel
University of Michigan, Ann Arbor
amangoel@umich.edu

Karem A. Sakallah
University of Michigan, Ann Arbor
karem@umich.edu

Abstract—Lamport’s celebrated Paxos consensus protocol is generally viewed as a complex hard-to-understand algorithm. Notwithstanding its complexity, in this paper, we take a step towards automatically proving the safety of Paxos by taking advantage of three structural features in its specification: spatial regularity in its unordered domains, temporal regularity in its totally-ordered domain, and its hierarchical composition. By carefully integrating these structural features in IC3PO, a novel model checking algorithm, we were able to infer an inductive invariant that identically matches the human-written one previously derived with significant manual effort using interactive theorem proving. While various attempts have been made to verify different versions of Paxos, to the best of our knowledge, this is the first demonstration of an automatically-inferred inductive invariant for Lamport’s original Paxos specification. We note that these structural features are not specific to Paxos and that IC3PO can serve as an automatic general-purpose protocol verification tool.

Index Terms—Distributed protocols, incremental induction, inductive invariant, invariant inference, model checking, Paxos.

I. INTRODUCTION

In this paper, we focus on proving the safety of distributed protocols like Paxos [1], [2] which form the basis for implementing many efficient and highly fault-tolerant distributed services [3]–[5]. Developed by Lamport, the Paxos consensus protocol allows a set of processes to communicate with each other by exchanging messages and reach agreement on a single value. Verifying the correctness of such a concurrent system requires the derivation of a quantified inductive invariant that, together with the protocol specification, acts as an inductive proof of its safety under all possible system behaviors.

Several manual or semi-automatic verification techniques based on interactive theorem proving [6]–[9] have been proposed to derive a safety proof for Paxos. Chand et al. [10] formally verified the TLA+ [11] specification of Paxos by manually deriving a proof using the TLAPS proof assistant [7]. Padon et al. [12] used the Ivy [13] verifier, which requires a user to manually refine automatically-generated counterexamples-to-induction, to obtain an inductive invariant for a simplified version of Paxos in the decidable EPR fragment [14] of first-order logic. The approaches in [15]–[19] are examples of manually-derived refinement proofs [20]–[23] that show how a low-level implementation refines a high-level specification. All these methods, however, require a detailed understanding of the intricate inner workings of the protocol and entail significant manual effort to guide proof development.

In contrast, we propose an approach, implemented in the IC3PO protocol verifier, to automatically infer the required inductive invariant for an unbounded distributed protocol by adding three simple extensions to the finite-domain IC3/PDR [24], [25] incremental induction algorithm for model checking [26]. Symmetry boosting, introduced in [27], takes advantage of a protocol’s spatial regularity to automatically infer quantified strengthening assertions that reflect the protocol’s structural symmetries. This paper describes range boosting and hierarchical strengthening which take advantage, respectively, of a protocol’s temporal regularity and hierarchical structure, and demonstrates how IC3PO was used to automatically obtain an inductive invariant for Paxos using the four-level hierarchy shown in Figure 1.

Our main contributions are:

- A range boosting technique that extends incremental induction to utilize the temporal regularity in totally-ordered domains, and thus, enables automatic invariant inference for protocols with even infinite-state processes.
- A hierarchical strengthening approach to derive the required inductive invariant in a top-down step-wise procedure for hierarchically-specified distributed protocols through incremental induction extended with symmetry and range boosting, by automatically verifying high-level abstractions first and using invariants of these higher-level specifications.

Fig. 1: Hierarchical strengthening of Paxos and its variants. Each level uses all strengthening assertions above that level as input, and outputs the required remaining assertions, altogether inferring the inductive invariant at each level.
level abstractions as strengthening assertions to derive the inductive invariant for the detailed lower-level protocol. Safety verification of Lamport’s Paxos algorithm, both single- and multi-decreed Paxos, through the derivation of a compact, human-readable inductive proof that is automatically inferred using IC3PO, resulting in a drastic reduction in verification effort compared to previous approaches [16], [28], [29].

The paper is structured as follows: §II presents preliminaries. §III and §IV describe range boosting and hierarchical strengthening. §V details the four-level hierarchy we used to prove Paxos and §VI is a record of the IC3PO run showing the actual assertions it inferred at each level of the hierarchy. §VII discusses some of the features and interesting details on this automatically-generated proof. Experimental comparisons with other approaches are provided in §VIII and the paper concludes with a brief survey of related work in §IX and a discussion of future directions in §X.

II. PRELIMINARIES

A. Notation

We will use Init, Next, and Safety to denote the quantified formulas that specify, respectively, a protocol’s initial states, its transition relation, and the safety property that is required to hold on all reachable states. We use primes (e.g., \( \varphi' \)) to represent a formula after a single transition step. The notation \( V!A \) (resp. \( S!A \), \( I!A \), and \( P!A \)) means that assertion \( A \) was inferred by IC3PO for the Voting (resp. SimplePaxos, ImplicitPaxos, and Paxos) protocol.

As an example, consider a protocol \( P \) with two sorts, a symmetric sort \( \text{aSort} \) and a totally-ordered sort \( \text{bSort} \), along with relations \( p(\text{aSort}, \text{bSort}) \) and \( q(\text{bSort}) \) defined on these sorts. Viewed as a parameterized system \( P(\text{aSort}, \text{bSort}) \), we can specify its finite instance \( P(3, 4) \) as:

\[
P(3, 4) : \quad \text{aSort}_3 \triangleq \{a_1, a_2, a_3\} \quad \text{bSort}_4 \triangleq [b_{\text{min}}, b_1, b_2, b_{\text{max}}]
\]

where \( \text{aSort}_3 \) represents the finite symmetric sort of this instance defined as a set of arbitrarily-named distinct constants, while the finite totally-ordered sort \( \text{bSort}_4 \) is composed of a list of ordered constants, i.e., \( b_{\text{min}} < b_1 < b_2 < b_{\text{max}} \). This instance can be encoded using twelve \( p \) and four \( q \) BOOLEAN state variables. A state of this instance corresponds to a complete assignment to these 16 state variables, with a total state-space size of \( 2^{16} \). We will use Next instead of Next to denote the transition relation of the finite instance.

B. Clause Boosting and Quantifier Inference

The basic framework for inferring the quantified assertions required to prove protocol safety is described in [27]. It extends the finite IC3PDR incremental induction algorithm by boosting its clause learning during the 1-step backward reachability checks performed through Satisfiability Modulo Theories (SMT) [30] solving. Specifically, a clause \( \varphi \) is learned in (and refines) frame \( F_i \) if the 1-step query \( \psi_i := F_{i-1} \land \text{Next} \land [\neg \varphi] \) is unsatisfiable. This means that cube \( \neg \varphi \) in frame \( F_i \) is unreachable from frame \( F_{i-1} \). Boosting refers to: a) “growing” \( \varphi \) to a set of clauses that also satisfy this unreachability constraint from frame \( F_{i-1} \), and b) refining the frame \( F_i \) with the entire clause set instead of just \( \varphi \). Such boosting accelerates the convergence of incremental induction but, more importantly, makes it possible, under some regularity assumptions, to represent this set of clauses by a single logically-equivalent quantified clause \( \Phi \) and is the key to generalizing the results of such finite analysis to unbounded domains.

C. Symmetric Boosting and Quantifier Inference

Protocols that are strictly specified in terms of symmetric sorts can be characterized as having spatial regularity. For example, the constants in a sort representing a finite set of \( k \) identical processes are essentially indistinguishable replicas that can be permuted arbitrarily without changing the protocol behavior. A learned clause \( \varphi \) parameterized by the constants of such a sort can be boosted by permuting its constants in all possible \( k! \) ways yielding a set of symmetrically-equivalent clauses, i.e., its symmetry orbit \( \varphi_{\text{Symk}} \) under the full symmetric group \( \text{Sym}_k \). By construction, all clauses in \( \varphi \)’s orbit automatically satisfy the unreachability constraint without the need to perform additional 1-step queries. Furthermore, the quantified clause \( \Phi \) that encodes \( \varphi \)’s orbit is algorithmically constructed by a syntactic analysis of \( \varphi \)’s structure, and can involve complex universal and existential quantifier alternations over both state and non-state (auxiliary) variables. The reader is referred to [27], [31] for the complete details of the connection between symmetry and quantification and the procedure for quantifier inference.

D. Finite Convergence

When a boosted finite incremental induction run terminates, it either produces a finite counterexample demonstrating that the specified safety property fails, or produces a set of quantified assertions \( A_1, \ldots, A_n \) that yield the inductive invariant \( \text{inv} = \text{Safety} \land A_1 \land \cdots \land A_n \) proving safety for the given finite size. At this point, an algorithmic finite convergence procedure is invoked to check if the current instance size has captured all possible protocol behaviors and, if not, to systematically increase the finite instance size until protocol behavior saturates and the cutoff size is reached [32]–[36].

III. RANGE BOOSTING

Clause boosting is not limited to clauses that are parameterized by the constants of symmetric sorts, and can be extended to clauses whose literals depend on the constants of totally-ordered sorts such as ballot, round, epoch, etc., that are used to model the temporal order of events in a distributed protocol. However, the boosting procedure for such clauses differs from symmetric boosting in two ways: a) the ordering relation between totally-ordered constants must be explicitly preserved, and b) adherence of a boosted clause to the unreachability
constraint is not guaranteed and must be explicitly checked with a 1-step backward reachability query.

We extended IC3PO with a range boosting procedure that complements its symmetry boosting mechanism, allowing it to transparently handle protocols with both symmetric and totally-ordered sorts.

Let \( \varphi \) be a clause that is parameterized by totally-ordered constants and let \( \varphi^{\text{Ordered}} \) denote those variants of \( \varphi \) that are obtained by ordering-compliant permutations of its constants. Clause \( \varphi \) is boosted by making 1-step backward reachability queries on \( \varphi^{\text{Ordered}} \) to identify its safe subset \( \varphi^{\text{Safe}} \), i.e., those variants that satisfy the unreachability constraint.

For example, consider the following clause \( \varphi_1 \) defined on the finite instance \( P(3, 4) \) from (1):

\[
\varphi_1 = p(a_1, b_1) \lor q(b_2)
\]  

(2)

Since \( \varphi_1 \) contains two ordered constants \((b_1, b_2)\), it has six ordering-compliant variants \((b_{\text{min}}, b_1, b_{\text{min}}, b_2), (b_{\text{min}}, b_2, b_{\text{min}}, b_1), (b_{\text{min}}, b_1), (b_{\text{max}}, b_1), (b_1, b_{\text{max}}), (b_2, b_{\text{max}})\). However only three of these variants end up satisfying the unreachability constraint yielding the following safe subset of \( \varphi_1^{\text{Ordered}} \):

\[
\begin{align*}
\varphi_1^{\text{Safe}} &= [ p(a_1, b_1) \lor q(b_2) ] \\
&\quad \lor [ p(a_1, b_1) \lor q(b_{\text{max}}) ] \\
&\quad \lor [ p(a_1, b_2) \lor q(b_{\text{max}}) ]
\end{align*}
\]  

(3)

The inferred quantified clause that encodes these three clauses is now constructed using two universally-quantified variables \( X_1, X_2 \in b\text{Sort}_4 \) that replace \( b_1 \) and \( b_2 \) in \( \varphi_1 \) and expressed as an implication whose antecedent specifies a constraint over the ordered “range” \( b_{\text{min}} < X_1 < X_2 \) that must be satisfied by the quantified variables:

\[
\Phi_1 = \forall X_1, X_2 \in b\text{Sort}_4 : (b_{\text{min}} < X_1) \land (X_1 < X_2) \rightarrow [ p(a_1, X_1) \lor q(X_2) ]
\]  

(4)

In general, a clause that is parameterized by \( k \) constants from a totally-ordered domain whose size is greater than \( k \) can be range-boosted and encoded by a universally-quantified predicate with \( k \) variables which is expressed as an implication whose antecedent is a range constraint that evaluates to true for just those combinations of the \( k \) variables that correspond to safe variants of \( \varphi \).

This procedure extends easily to the case of multiple totally-ordered domains as well, allowing range boosting to be performed independently for each such domain in any order since constants from different domains do not interfere with each other.

IV. HIERARCHICAL STRENGTHENING

As advocated in [37], hierarchical structuring is an effective way to manage complexity during manual proof development. It can also be easily incorporated in the IC3PO style of invariant generation based on symmetry and range boosting.

Given a low-level specification \( L \) that implements a high-level specification \( H \), i.e., \( L \prec H \), hierarchical strengthen-
Fig. 2: Lamport’s Voting protocol in pretty-printed TLA+

Viewed as a parameterized system, the template of the Voting protocol is Voting(value, acceptor, quorum, ballot). Its finite instance:

Voting(2, 3, 3, 4):

\[\begin{align*}
\text{value}_2 & \triangleq \{v_1, v_2\} \\
\text{acceptors} & \triangleq \{a_1, a_2, a_3\} \\
\text{quorums}_3 & \triangleq \{q_{12}:\{a_1, a_2\}, q_{13}:\{a_1, a_3\}, q_{23}:\{a_2, a_3\}\} \\
\text{ballots}_4 & \triangleq \{b_{\text{min}}, b_1, b_2, b_{\max}\}
\end{align*}\]

has three finite symmetric sorts named value\(_2\), acceptor\(_3\) and quorum\(_3\), defined as sets of arbitrarily-named distinct constants, while the finite totally-ordered sort ballots\(_4\) is composed of a list of ordered constants, i.e., \(b_{\text{min}} < b_1 < b_2 < b_{\max}\), where \(b_{\text{min}} = -1\) since \(-1\) is the “minimum” ballot number. The constants of the quorum\(_3\) sort are subsets of the acceptor\(_3\) sort and are named to reflect

Fig. 3: Lamport’s Paxos protocol in pretty-printed TLA+
their symmetric dependence on the acceptor\textsubscript{3} sort. This instance has 24 votes state variables that return a BOOLEAN and 3 maxBal state variables that return a ballot number in ballot\textsubscript{4}. A state of this instance corresponds to a complete assignment to these 27 state variables.

B. Lamport’s Paxos Protocol

Figure 3 presents the TLA+ description of Lamport’s Paxos protocol [39], which is a specification of the Paxos consensus algorithm [1], [2]. Paxos implements Voting through the refinement mapping \([\text{votes} \leftarrow \text{msg}2b, \text{maxBal} \leftarrow \text{maxBal}]\), where acceptors now communicate with each other through distributed message passing. State variables msg\textsubscript{1}a, msg\textsubscript{1}b, msg\textsubscript{2}a, and msg\textsubscript{2}b are used to model the set of different messages that can be sent in the protocol, corresponding to actions Phase\textsubscript{1}a, Phase\textsubscript{1}b, Phase\textsubscript{2}a, and Phase\textsubscript{2}b respectively. The pair \((\text{maxBal}(a), \text{maxVal}(a))\) is the vote with the largest ballot number cast by acceptor \(a\). The ballot \(b\) leader can send a msg\textsubscript{1}a(b) by performing the action Phase\textsubscript{1}a(b). Phase\textsubscript{1}b(a, b) implements the IncreaseMaxBal action from Voting, where after receiving msg\textsubscript{1}a(b), acceptor \(a\) sends msg\textsubscript{1}b to the ballot \(b\) leader containing the values of maxVBal(a) and maxVal(a). In the Phase\textsubscript{2}a(b, v) action, the ballot \(b\) leader sends msg\textsubscript{2}a asking the acceptors to vote for a value \(v\) that is safe at ballot number \(b\). Its enabling condition isSafeAtPaxos(b, v) checks the enabling condition isSafeAt(b, v) from Voting. Phase\textsubscript{2}b implements the VoteFor action in Voting, and enables acceptor \(a\) to vote for value \(v\) in ballot number \(b\). We refer the reader to [40] for a detailed explanation to understand the internals of Paxos.

Represented as a parameterized system Paxos(value, acceptor, quorum, ballot), its finite instance Paxos(2, 3, 3, 4) has 132 BOOLEAN state variables, 6 state variables that return a ballot number in ballot\textsubscript{4}, and 3 state variables that return a value in value\textsubscript{2}.

C. Intermediate Levels between Voting and Paxos

We introduced two intermediate levels, SimplePaxos and ImplicitPaxos, between Voting and Paxos. These intermediate levels are abstractions of Paxos, inspired from the already-existing literature [12], [41]–[44]. ImplicitPaxos is inspired from the specification of Generalized Paxos by Lamport [41] and uses a commonly-used encoding transformation, as utilized in [12], [43], [44]. Instead of explicitly keeping a track of maxVBal(a) and maxVal(a), ImplicitPaxos abstracts them away and implicitly computes their respective values using the history of all votes cast by the acceptor \(a\), i.e., using the history of msg\textsubscript{2}b from acceptor \(a\), by modifying the Phase\textsubscript{1}b(a, b) action (line 11 in Figure 3) as shown in Figure 4.

SimplePaxos further simplifies ImplicitPaxos and eliminates tracking of the maximum ballot (and the corresponding value) in which an acceptor voted from msg\textsubscript{1}b completely, i.e., the last two arguments of msg\textsubscript{1}b are abstracted away. Instead, the history of all votes cast is used to describe how new votes are cast. This is done by replacing the definition

\[
\text{maxVal}(a) = \text{maxVal}(a) \cup \text{msg}2b(a, b)
\]

showSafeAtPaxos (line 8 in Figure 3) with its simplified form, expressed using msg\textsubscript{2}b as shown in Figure 5.

VI. HIERARCHICAL VERIFICATION OF PAXOS

Using the 4-level hierarchy Paxos ≺ ImplicitPaxos ≺ SimplePaxos ≺ Voting, this section is a “log” of how IC3PO automatically derived the required strengthening assertions that established the safety of Paxos.

A. Proving Voting

Using instance Voting(2, 3, 3, 4), IC3PO proved the safety of Voting by automatically deriving the inductive invariant \(V!\text{live} \equiv V!\text{Safety} \land V!A_1 \land V!A_2\) where

\[
V!A_1 = \forall A \in \text{acceptor}, B \in \text{ballot}, V \in \text{value} : \text{votes}(A, B, V) \rightarrow \text{isSafeAt}(B, V)
\]

\[
V!A_2 = \forall A \in \text{acceptor}, B \in \text{ballot}, V_1, V_2 \in \text{value} : \text{chosenAt}(B, V_1) \wedge \text{votes}(A, B, V_2) \rightarrow (V_1 = V_2)
\]

In words, these two strengthening assertions mean:

\begin{itemize}
  \item \(A_1\): If an acceptor voted for value \(V\) in ballot number \(B\), then \(V\) is safe at \(B\).
  \item \(A_2\): If value \(V_1\) is chosen at ballot \(B\), then no acceptor can vote for a value different than \(V_1\) in \(B\).
\end{itemize}

B. Proving SimplePaxos

Using the refinement mapping \([\text{votes} \leftarrow \text{msg}2b, \text{maxBal} \leftarrow \text{maxBal}]\), IC3PO transformed \(V!A_1\) and \(V!A_2\) to the follow-
ing corresponding versions for SimplePaxos:

\[ S!A_1 = \forall A \in \text{acceptor}, B \in \text{ballot}, V \in \text{value} : \]
\[ \text{msg}2b(A, B, V) \rightarrow \text{isSafeAt}(B, V) \]
\[ S!A_2 = \forall A \in \text{acceptor}, B \in \text{ballot}, V_1, V_2 \in \text{value} : \]
\[ \text{chosenAt}(B, V_1) \land \text{msg}2b(A, B, V_2) \rightarrow (V_1 = V_2) \]

These two assertions, passed down from the proof of Voting, represented a strengthening of the safety property of SimplePaxos that allowed IC3PO to prove it with the inductive invariant \( S!\text{inv} \triangleq S!\text{Safety} \land \bigwedge_{1 \leq i \leq 6} S!A_i \), where

\[ S!A_3 = \forall B \in \text{ballot}, V \in \text{value} : \]
\[ \text{msg}2a(B, V) \rightarrow \text{isSafeAt}(B, V) \]
\[ S!A_4 = \forall B \in \text{ballot}, V_1, V_2 \in \text{value} : \]
\[ \text{msg}2a(B, V_1) \land \text{msg}2a(B, V_2) \rightarrow (V_1 = V_2) \]
\[ S!A_5 = \forall A \in \text{acceptor}, B \in \text{ballot}, V \in \text{value} : \]
\[ \text{msg}2b(A, B, V) \rightarrow \text{msg}2a(B, V) \]
\[ S!A_6 = \forall A \in \text{acceptor}, B \in \text{ballot} : \]
\[ \text{msg}1b(A, B) \rightarrow \text{maxBal}(A) \geq B \]

are four additional automatically-generated strengthening assertions that express the following facts about SimplePaxos:

A3: If ballot B leader sends a 2a message for value V, then V is safe at B.
A4: A ballot leader can send 2a messages only for a unique value.
A5: If an acceptor voted for a value in ballot number B, then there is a 2a message for that value at B.
A6: If an acceptor has sent a 1b message at a ballot number B, then its maxBal is at least as high as B.

C. Proving ImplicitPaxos

All variables from SimplePaxos refine to ImplicitPaxos as is, except for msg1b that adds explicit tracking of the maximum vote voted by an acceptor in ImplicitPaxos. Assertions \( S!A_1 \) to \( S!A_5 \) map to \( !1A_1 \) to \( !1A_5 \) in ImplicitPaxos as is, while \( S!A_6 \) maps as:

\[ !1A_6 = \forall A \in \text{acceptor}, B, B_{max} \in \text{ballot}, V_{max} \in \text{value} : \]
\[ \text{msg}1b(A, B, B_{max}, V_{max}) \rightarrow \text{maxBal}(A) \geq B \]

These six assertions, passed down from the proof of SimplePaxos, represented a strengthening of the safety property of ImplicitPaxos that allowed IC3PO to prove it with the inductive invariant \( I!\text{inv} \triangleq I!\text{Safety} \land \bigwedge_{1 \leq i \leq 8} I!A_i \), where

\[ I!A_7 = \forall A \in \text{acceptor}, B, B_{max} \in \text{ballot}, V_{max} \in \text{value} : \]
\[ [(B > -1) \land (B_{max} > -1) \land \text{msg}1b(A, B, B_{max}, V_{max})] \]
\[ \rightarrow \text{msg}2b(A, B_{max}, V_{max}) \]
\[ I!A_8 = \forall A \in \text{acceptor}, B, B_{mid}, B_{max} \in \text{ballot}, \]
\[ V, V_{max} \in \text{value} : \]
\[ [(B > B_{mid}) \land (B_{mid} > B_{max}) \land \text{msg}1b(A, B, B_{max}, V_{max})] \]
\[ \rightarrow \neg \text{msg}2b(A, B_{mid}, V) \]

are two additional automatically-generated strengthening assertions that express the following facts about ImplicitPaxos:

A7: If an acceptor issued a 1b message at ballot number B with the maximum vote \( \langle B_{max}, V_{max} \rangle \), and both B and \( B_{max} \) are higher than \(-1\), then the acceptor has voted for value \( V_{max} \) in ballot \( B_{max} \).
A8: If an acceptor issued a 1b message at ballot number B with the maximum vote \( \langle B_{max}, V_{max} \rangle \), then the acceptor cannot have voted in any ballot number strictly between \( B_{max} \) and B.

D. Proving Paxos

All variables from ImplicitPaxos refine to Paxos trivially, mapping \( !1A_1, \ldots, !1A_8 \) to \( P!A_1, \ldots, P!A_6 \) in Paxos as is. These eight assertions, passed down from the proof of ImplicitPaxos, represented a strengthening of the safety property of Paxos that allowed IC3PO to prove it with the inductive invariant \( P!\text{inv} \triangleq P!\text{Safety} \land \bigwedge_{1 \leq i \leq 11} P!A_i \), where

\[ P!A_9 = \forall A \in \text{acceptor} : \text{maxVBal}(A) \leq \text{maxBal}(A) \]
\[ P!A_{10} = \forall A \in \text{acceptor}, B \in \text{ballot}, V \in \text{value} : \]
\[ \text{msg}2b(A, B, V) \rightarrow \text{maxVBal}(A) \geq B \]
\[ P!A_{11} = \forall A \in \text{acceptor} : \text{maxVBal}(A) \geq 1 \]
\[ \rightarrow \text{msg}2b(A, \text{maxVBal}(A), \text{maxVal}(A)) \]

are three additional automatically-generated strengthening assertions that express the following facts about Paxos:

A9: \( \text{maxVBal} \) of an acceptor is less than or equal to its \text{maxBal}.
A10: If an acceptor voted in a ballot number B, then its \text{maxVBal} is at least as high as B.
A11: If acceptor A has its \text{maxVBal} higher than \(-1\), then A has already cast a vote \( \langle \text{maxVBal}(A), \text{maxVal}(A) \rangle \).

VII. DISCUSSION

This section provides a discussion about certain key points and features about the Paxos proof from Section VI.

A. Comparison against Human-written Invariants

Optionally, the inductive invariant \( I!\text{inv} \) can be minimized to derive a subsumption-free and closed set of invariants, which removes \( A_1 \) and \( A_2 \) that are subsumed by the conjunction \( A_3 \land A_4 \land A_5 \). After this minimization, the inductive invariant of Paxos matches identically with the manually-written and TLAPS-checked inductive invariant from [28], guaranteeing its correctness. Similarly, the inductive invariant of Voting, i.e., \( V!\text{inv} \), matches directly with the manually-written and TLAPS-checked inductive invariant from [45].

B. Benefits of Range Boosting

Assertions \( A_6 \) to \( A_{11} \) express conditions defined over ordered ranges in the \text{infinite} totally-ordered \text{ballot} domain. Inferring such invariants automatically through IC3PO becomes possible through range boosting (Section III), that extends incremental induction with the knowledge of temporal regularity over totally-ordered domains by learning quantified clauses over ordered ranges.
C. Protocol's Formula Structure

Note that $A_1$ to $A_3$ use definitions $\text{isSafeAt}$ and $\text{chosenAt}$, which implicitly enables IC3PO to incorporate learning with complex quantifier alternations. Inspired from previous works on the importance of using derived/ghost variables [36], [46], [47], IC3PO utilizes the formula structure of the protocol’s transition relation in a unique manner, by incorporating definitions in the protocol specification as auxiliary non-state variables during reachability analysis, described in detail in [27]. This provides a simple and inexpensive procedure to incorporate clause learning with complex quantifier alternations.

D. Decidability

Protocol specifications at each of the four levels include quantifier alternation cycles that make unbounded SMT reasoning fall into the undecidable fragment of first-order logic. Unsurprisingly, previous works that rely on unbounded SMT reasoning, like SWISS [48], fol-ic3 [49], DistAI [50], I4 [51], and UPDR [52], struggle with verifying Lamport’s Paxos.

IC3PO, on the other hand, performs incremental induction and finite convergence over finite protocol instances using finite-domain reasoning that is always decidable.

E. Why a Four-Level Hierarchy?

The original Paxos specification is composed of a two-level hierarchy $\text{Paxos} \not \stackrel{\text{Voting}}{\preceq} \text{Voting}$. Given the two strengthening assertions $A_1$ and $A_2$ from $\text{Voting}$, inferring the remaining nine assertions for $\text{Paxos}$ directly in one step of hierarchical strengthening is difficult, since these two specifications are too far apart to be proved directly. IC3PO struggled with the large state space of $\text{Paxos}$ and learnt too many weak clauses involving $\text{msg1b}$, $\text{maxVBal}$ and $\text{maxVal}$, eventually running out of memory due to invariant inference getting confused with several counterexamples-to-induction. Table I compares the state-space size of protocol instances at each of the four hierarchical levels. Even though $2^{147}$ is not huge, especially with respect to hardware verification problems [53–55], $\text{Paxos}$ has a dense state-transition graph where state-transitions are tightly coupled with high in- and out-degree, making the problem difficult for automatic invariant inference with incremental induction based model checking.

Adding $\text{ImplicitPaxos}$ reduced the complexity in $\text{Paxos}$ by abstracting away $\text{maxVBal}$ and $\text{maxVal}$. Still, scalability remained a challenge due to $\text{msg1b}$, that contributed to 96 out of 147 state bits in $\text{Paxos}(2, 3, 3, 4)$. Adding another level, i.e., $\text{SimplePaxos}$, removed 84 out of these 96 state bits by abstracting away explicit tracking of the maximum vote of an acceptor from $\text{msg1b}$. When compared against $\text{Paxos}$, $\text{SimplePaxos}$ is significantly simpler, with a total state-space size to be just $2^{54}$ for its finite instance $\text{SimplePaxos}(2, 3, 3, 4)$, which led IC3PO to successfully prove $\text{Paxos}$ automatically using the four-level hierarchy.

F. Extension to $\text{MultiPaxos}$ and $\text{FlexiblePaxos}$

Till now, by $\text{Paxos}$ we meant single-decree $\text{Paxos}$ which is the core consensus algorithm underlying the complete $\text{Paxos}$ state-machine replication protocol [1], [2], commonly referred to as $\text{MultiPaxos}$ [43]. In $\text{MultiPaxos}$, a sequence of instances execute single-decree $\text{Paxos}$ such that the value chosen in the $i^{\text{th}}$ instance becomes the $i^{\text{th}}$ command executed by the replicated state machine. Additionally, if the leader is relatively stable, $\text{Phase1}$ becomes unnecessary and is skipped, reducing the failure-free message delay from 4 delays to 2 delays.

Mapping each of the assertions $A_1, \ldots, A_{11}$ to $\text{MultiPaxos}$ is trivial, and simply adds the corresponding instance as an additional universally-quantified argument, e.g., $A_{11}$ maps as:

$$M!A_{11} = \forall A \in \text{acceptor}, i \in \text{instances}: \maxVBal(A, I) > -1 \rightarrow \text{msg2b}(A, I, \maxVBal(A, I), \maxVal(A, I))$$

Unsurprisingly, the 11 strengthening assertions, passed down from the proof of $\text{Paxos}$, together with the safety property of $\text{MultiPaxos}$, allowed IC3PO to trivially prove it with no additional strengthening assertions needed, meaning $M!\text{Safety} \land \bigwedge_{1 \leq i \leq 11} M!A_i$ is already an inductive invariant of $\text{MultiPaxos}$. As described in previous works [1], [2], [6], [10], the crux of proving the safety of $\text{MultiPaxos}$ is based on proving single-decree $\text{Paxos}$ since each consensus instance participates independently without any interference from other instances. Our experiments validated this further.

Similarly, we also tried another Paxos variant called $\text{FlexiblePaxos}$ [56], which also verifies trivially with the same inductive invariant, i.e., with no additional strengthening assertions needed.

### VIII. Experiments

IC3PO [57] currently accepts protocol descriptions in the Ivy language [13] and uses the Ivy compiler to extract a logical formulation of the protocol in a SMT-LIB [30] compatible format. To get an idea on the effectiveness of hierarchical strengthening, we also evaluated automatically deriving inductive proofs for EPR variants of Paxos from [12] without any hierarchical strengthening. These specifications describe Paxos in the EPR fragment [14] of first-order logic and also incorporate simplifications equivalent to the ones described for $\text{SimplePaxos}$ in Section V-C. We performed a detailed comparison against other state-of-the-art techniques for automatically verifying distributed protocols:

- SWISS [48] uses SMT solving to derive an inductive invariant by performing an enumerative search in an optimized and bounded invariant search space.
- fol-ic3 [49], implemented in mypy [58], extends IC3 with a separators-based technique that performs enumer-
Table II: Comparison of IC3PO against other state-of-the-art verifiers

<table>
<thead>
<tr>
<th>Protocol</th>
<th>S.A.</th>
<th>IC3PO</th>
<th>SWISS</th>
<th>fol-ic3</th>
<th>DistAI</th>
<th>I4</th>
<th>UPDR</th>
<th>IC3PO Human</th>
<th>IC3PO I4</th>
</tr>
</thead>
<tbody>
<tr>
<td>epr-paxos</td>
<td>⊗</td>
<td>568</td>
<td>1595</td>
<td>timeout</td>
<td>error</td>
<td>memout</td>
<td>timeout</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>epr-flexible_paxos</td>
<td>⊗</td>
<td>561</td>
<td>18232</td>
<td>timeout</td>
<td>error</td>
<td>memout</td>
<td>failure</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>epr-multi_paxos</td>
<td>⊗</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
<td>memout</td>
<td>timeout</td>
<td>timeout</td>
<td>13</td>
<td>12</td>
</tr>
</tbody>
</table>

**Table II: Comparison of IC3PO against other state-of-the-art verifiers**

ORIGINAL problems employ hierarchical strengthening (as detailed in Section VI), while EPR problems do not.

Column 2 (labeled S.A.) lists strengthening assertions added through hierarchical strengthening to the safety property (⊗ means none). Columns 3-8 (labeled Time) compare the runtime in seconds. For failed SWISS runs, we include the runtime from [48] (indicated with *).

Columns 9-10 (labeled Inv) compare number of assertions in the inductive invariant between IC3PO (with subsumption checking and minimization) and human-written proofs.

Columns 11-12 (labeled SMT) compare total number of SMT queries made by IC3PO versus I4 (until failure for unsuccessful runs).

B. Discussion

Effect of hierarchical strengthening: Comparing EPR versus ORIGINAL shows the advantages offered by hierarchical strengthening. Even though IC3PO was able to automatically verify EPR versions of single-decree Paxos and flexible Paxos from [12], none of the tools were able to automatically verify the EPR version of multi-decree Paxos. ORIGINAL variants, on the other hand, employed hierarchical strengthening which allowed IC3PO to verify Lamport’s Paxos automatically and efficiently by using the protocol’s hierarchical structure.

Comparison against other verifiers: DistAI failed on all problems due to unsupported constructs and parsing errors. I4 and UPDR (as well as DistAI) are limited to generating only universally-quantified invariants over state variables, and hence, were unable to solve any problem. While both IC3PO and I4 use incremental induction over a finite protocol instance, the number of SMT queries made by I4 grows drastically, indicating the benefits offered by symmetry and range boosting employed in IC3PO. fol-ic3 also fails on all problems, showing limited scalability of its enumeration-based separators technique operating directly in the unbounded domain. For SWISS, we weren’t able to replicate results for EPR problems as reported in [48] using our experimental setup. Nevertheless, SWISS showed limited capabilities for solving ORIGINAL problems.

Comparison against human-written invariants: As evident from A1 to A11 in Section VI, IC3PO generated concise, human-readable inductive invariants. In fact, every invariant of Paxos written manually by Lamport et al. (as detailed in [28], [39]) had a corresponding equivalent invariant in the inductive proof automatically generated with IC3PO. In contrast, deriving such invariants manually, even in the presence of a hierarchical structure, is a tedious and error-prone process that demands deep domain expertise [12], [16], [28], [29].

Overall, the evaluation confirms our main hypothesis, that it is possible to utilize the regularity and hierarchical structure in complex distributed protocols, like in Paxos, to scale automatic verification beyond the current state-of-the-art.
IX. RELATED WORK

Introduced by Lamport, TLA+ is a widely-adopted language for the specification and verification of distributed protocols [65], [66]. The TLA+ toolbox [67] provides the TLC model checker, which is primarily used as a debugging tool for verifying small finite protocol instances [68], and not as a tool for inferring inductive invariants. The TLAPS proof assistant [7], [8] allows checking proofs manually written in TLA+, and has been used to verify several distributed protocols, including variants of Paxos [10], [15].

The derivation of inductive invariants for distributed protocols continues to be mostly carried out through refinement proofs using interactive theorem proving [13], [16], [17], [19], [69]–[72], which demands significant manual effort and profound domain expertise. The first attempts at automatically deriving quantified invariants were reported in [32], [33], using invisible invariants. The intuition underlying this method was the assumption that the system is "sufficiently symmetric," and that its behavior can be captured by any m-subset of its processes as a universally-quantified invariant. However, universally-quantified invariants are not guaranteed to be inductive or to imply the safety property. Spatial regularity was further explored in [73]–[78] to reduce the verification of an n-process system to that of a quotient system at a small cutoff size.

Notwithstanding the undecidability result of Apt and Kozen [79], many efforts to automatically infer quantified inductive invariants have been reported with the pace increasing in recent years [48], [50]–[52], [80]–[82]. Verification of parameterized systems is further explored in [83]–[87]. However, unlike IC3PO, these methods generally do not scale to complex protocols like Lamport’s Paxos, since these methods rely heavily on unbounded reasoning and are limited to specifications in the EPR fragment of first-order logic.

Our technique builds on these works, with the capability to automatically infer the required quantified inductive invariant using the latest advancements in model checking, by extending our recent work [27] on symmetry boosting and finite convergence with range boosting and hierarchical strengthening.

X. CONCLUSIONS & FUTURE WORK

We proposed range boosting, a novel technique that extends the incremental induction algorithm to utilize the temporal regularity in distributed protocols through quantified reasoning over ordered ranges. We also presented hierarchical strengthening, a simple technique that utilizes the hierarchical structure of protocol specifications to enable automatic verification of complex distributed protocols with high scalability. Given the four-level hierarchy of the Paxos specification, we showed that these techniques, coupled with our recent work on symmetry boosting and finite convergence, provide, to our knowledge, the first demonstration of an automatically-inferred inductive invariant for the original Lamport’s Paxos algorithm.

While introducing SimplePaxos and ImplicitPaxos to get the four-level Paxos hierarchy was quite easy, these intermediate levels were still added manually. It is appealing to explore counterexample-guided abstraction-refinement (CEGAR) techniques [88], [89] to automatically identify these intermediate levels whenever needed to overcome complexity. Specifically, investigating how to leverage clause learning feedback from incomplete runs to identify bottlenecks in proof inference and utilizing this information to automatically abstract away irrelevant details from the low-level protocol can help in making the complete procedure automatic end-to-end. We leave this investigation as future work.

Exploring inference with existential quantifiers in range boosting can also be an interesting future direction, though intuitively, existential quantification over temporal behaviors looks unnecessary for proving safety properties. Future work also includes automatically inferring inductive proofs for other distributed protocols, such as Byzantine Paxos [15], Raft [90], etc., and exploring the verification of consensus algorithms in blockchain applications.

DATA AVAILABILITY STATEMENT AND ACKNOWLEDGMENTS

The software and data sets generated and analyzed during the current study, including all experimental data, evaluation scripts, and IC3PO source code are available at https://github.com/amank-goyal/fmcad2021exp.

We thank Leslie Lamport for the TLA+ video course [91], which shaped several ideas presented in this paper. We thank the developers of TLA+ [92], [93], Yices [63], Z3 [62], pySMT [94], and Ivy [13] for making their tools openly available. We also thank the reviewers for their valuable comments.

REFERENCES


