



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.

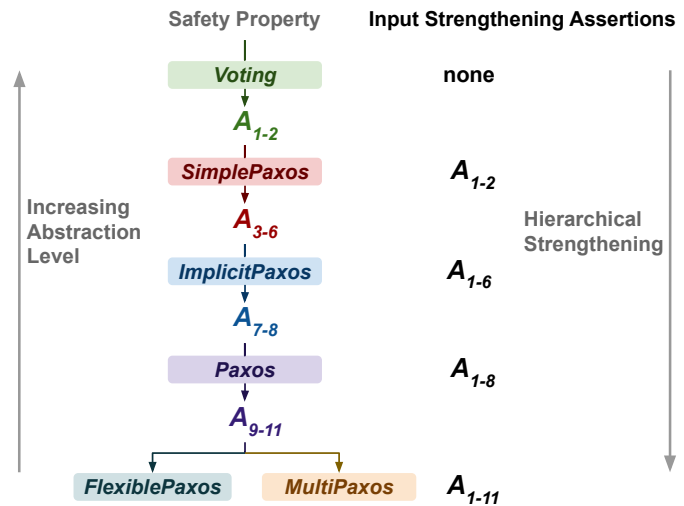


Fig. 1: Hierarchical strengthening of Paxos and its variants. Each level uses all strengthening assertions above that level as input, and outputs the required remaining assertions, altogether inferring the inductive invariant at each level.

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

$$\begin{aligned} \mathcal{P}(3, 4) : \quad & \text{aSort}_3 \triangleq \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\} \\ & \text{bSort}_4 \triangleq [\mathbf{b}_{\min}, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_{\max}] \end{aligned} \quad (1)$$

where 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 bSort_4 is composed of a list of ordered constants, i.e., $\mathbf{b}_{\min} < \mathbf{b}_1 < \mathbf{b}_2 < \mathbf{b}_{\max}$. 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 \widehat{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 IC3/PDR 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 φ is learned in (and refines) frame F_i if the 1-step query $\psi_i :=$

$F_{i-1} \wedge \widehat{Next} \wedge [\neg\varphi']$ is unsatisfiable. This means that cube $\neg\varphi$ in frame F_i is unreachable from frame F_{i-1} . Boosting refers to: a) “growing” φ 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 φ . 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* Φ 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 φ 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* φ^{Sym_k} under the full symmetric group Sym_k . By construction, all clauses in φ ’s orbit automatically satisfy the unreachability constraint without the need to perform additional 1-step queries. Furthermore, the quantified clause Φ that encodes φ ’s orbit is algorithmically constructed by a syntactic analysis of φ ’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, \dots, A_n that yield the inductive invariant $inv = Safety \wedge A_1 \wedge \dots \wedge A_n$ 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 φ be a clause that is parameterized by totally-ordered constants and let $\varphi^{Ordered}$ denote those variants of φ that are obtained by ordering-compliant permutations of its constants. Clause φ is boosted by making 1-step backward reachability queries on $\varphi^{Ordered}$ to identify its *safe* subset φ^{Safe} , i.e., those variants that satisfy the unreachability constraint.

For example, consider the following clause φ_1 defined on the finite instance $\mathcal{P}(3, 4)$ from (1):

$$\varphi_1 = p(\mathbf{a}_1, \mathbf{b}_1) \vee q(\mathbf{b}_2) \quad (2)$$

Since φ_1 contains two ordered constants $(\mathbf{b}_1, \mathbf{b}_2)$, it has six ordering-compliant variants $(\mathbf{b}_{\min}, \mathbf{b}_1)$, $(\mathbf{b}_{\min}, \mathbf{b}_2)$, $(\mathbf{b}_{\min}, \mathbf{b}_{\max})$, $(\mathbf{b}_1, \mathbf{b}_2)$, $(\mathbf{b}_1, \mathbf{b}_{\max})$, and $(\mathbf{b}_2, \mathbf{b}_{\max})$. However only three of these variants end up satisfying the unreachability constraint yielding the following safe subset of $\varphi_1^{Ordered}$:

$$\begin{aligned} \varphi_1^{Safe} = & [p(\mathbf{a}_1, \mathbf{b}_1) \vee q(\mathbf{b}_2)] \wedge \\ & [p(\mathbf{a}_1, \mathbf{b}_1) \vee q(\mathbf{b}_{\max})] \wedge \\ & [p(\mathbf{a}_1, \mathbf{b}_2) \vee q(\mathbf{b}_{\max})] \end{aligned} \quad (3)$$

The inferred quantified clause that encodes these three clauses is now constructed using two universally-quantified variables $X_1, X_2 \in \text{bSort}_4$ that replace \mathbf{b}_1 and \mathbf{b}_2 in φ_1 and expressed as an implication whose antecedent specifies a constraint over the ordered “range” $\mathbf{b}_{\min} < X_1 < X_2$ that must be satisfied by the quantified variables:

$$\begin{aligned} \Phi_1 = & \forall X_1, X_2 \in \text{bSort}_4 : \\ & (\mathbf{b}_{\min} < X_1) \wedge (X_1 < X_2) \rightarrow [p(\mathbf{a}_1, X_1) \vee q(X_2)] \end{aligned} \quad (4)$$

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 φ .

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-

ing starts by automatically deriving strengthening assertions $H!A^H$ that, together with the safety property $H!Safety$, proves the safety of H . It then maps and propagates $H!A^H$ to L , denoted as $L!A^H$, and proceeds to prove the strengthened property $L!Safety \wedge L!A^H$ in L by deriving any additional assertions $L!A^L$ needed to establish the safety of L . The underlying assumption in this procedure is that proving H is much easier than proving L directly, and that any assertions derived to prove H are also applicable, with suitable mapping, to L . The final inductive invariant that proves L will, thus, have the form $L!inv = (L!Safety \wedge L!A^H) \wedge L!A^L$ which can be interpreted as reducing the complexity of L ’s proof by strengthening its safety property with assertions derived for H .

Such strengthening can be extended to a k -level hierarchy $H \prec M_1 \prec \dots \prec M_{k-2} \prec L$, where M_1 to M_{k-2} are suitably-defined intermediate levels between H and L . This, in turn, allows single-level automatic verification techniques based on incremental induction, like IC3PO, to scale to complex protocols like *Paxos*, by step-wise verifying higher-level abstractions first and using their auto-generated proofs to incrementally build the proof for the lower-level protocol.

V. HIERARCHICAL SPECIFICATION OF PAXOS

This section describes in detail the multi-level hierarchical structure of the Paxos protocol, as shown earlier in Figure 1.

A. Lamport’s Voting Protocol

Figure 2 presents the TLA+ [11] description¹ of the *Voting* protocol [38], which is a very high-level abstraction of *Paxos* that formalizes the way Lamport first thought about the Paxos consensus algorithm without getting distracted by details introduced by having the processes communicate by messages. *Voting* has three unordered sorts named *value*, *acceptor* and *quorum*, and a totally-ordered sort named *ballot*. The protocol has two state symbols, *votes* and *maxBal* defined on these sorts that serve as the protocol’s state variables. $votes(a, b, v)$ is true iff an acceptor a has voted for value v in ballot number b . $maxBal(a)$ returns a ballot number such that acceptor a will never cast any further vote in a ballot numbered less than $maxBal(a)$. The global axiom (line 5) defines the elements of the *quorum* sort to be subsets of the *acceptor* sort and restricts them further by requiring them to be pair-wise non-disjoint. Lines 6-9 specify definitions *chosenAt*, *chosen*, *showsSafeAt*, and *isSafeAt*, which serve as auxiliary non-state variables. Protocol transitions are specified by the actions *IncreaseMaxBal* and *VoteFor* (lines 10-11), and lines 12-14 specify the protocol’s initial states, transition relation, and safety property.

¹Lamport’s TLA+ encoding uses sets to denote variables. For example in [38], $votes[a]$ represents the set of votes cast by acceptor a . Throughout this paper, we use an equivalent representation based on relations/functions to enable encoding for SMT solving. $\langle b, v \rangle \in votes[a]$ is equivalently encoded in relational form as $votes(a, b, v) = \top$.

```

1 CONSTANTS value, acceptor, quorum
2 ballot  $\triangleq$  Nat  $\cup$   $\{-1\}$ 
3 VARIABLES votes, maxBal
4 votes  $\in$  (acceptor  $\times$  ballot  $\times$  value)  $\rightarrow$  BOOLEAN
   maxBal  $\in$  acceptor  $\rightarrow$  ballot
5 ASSUME  $\wedge \forall Q \in$  quorum :  $Q \subseteq$  acceptor
    $\wedge \forall Q_1, Q_2 \in$  quorum :  $Q_1 \cap Q_2 \neq \{\}$ 
6 chosenAt( $b, v$ )  $\triangleq \exists Q \in$  quorum :  $\forall A \in Q : votes(A, b, v)$ 
7 chosen( $v$ )  $\triangleq \exists B \in$  ballot : chosenAt( $B, v$ )
8 showsSafeAt( $q, b, v$ )  $\triangleq$ 
    $\wedge \forall A \in q : maxBal(A) \geq b$ 
    $\wedge \exists C \in$  ballot :
      $\wedge (C < b)$ 
      $\wedge (C \neq -1) \rightarrow \exists A \in q : votes(A, C, v)$ 
      $\wedge \forall D \in$  ballot :
        $(C < D < b) \rightarrow$ 
          $\forall A \in Q : \forall V \in$  value :  $\neg votes(A, D, V)$ 
9 isSafeAt( $b, v$ )  $\triangleq \exists Q \in$  quorum : showsSafeAt( $Q, b, v$ )
10 IncreaseMaxBal( $a, b$ )  $\triangleq$ 
    $\wedge b \neq -1 \wedge b > maxBal(a)$ 
    $\wedge maxBal' = [maxBal \text{ EXCEPT } ![a] = b]$ 
    $\wedge$  UNCHANGED votes
11 VoteFor( $a, b, v$ )  $\triangleq$ 
    $\wedge b \neq -1 \wedge maxBal(a) \leq b$ 
    $\wedge \forall V \in$  value :  $\neg votes(a, b, V)$ 
    $\wedge \forall C \in$  acceptor :
      $(C \neq a) \rightarrow$ 
        $\forall V \in$  value : votes( $C, b, V$ )  $\rightarrow (V = v)$ 
    $\wedge isSafeAt(b, v)$ 
    $\wedge votes' = [votes \text{ EXCEPT } ![a, b, v] = \top]$ 
    $\wedge maxBal' = [maxBal \text{ EXCEPT } ![a] = b]$ 
12 Init  $\triangleq$ 
    $\wedge \forall A \in$  acceptor :  $B \in$  ballot :  $V \in$  value :  $\neg votes(A, B, V)$ 
    $\wedge \forall A \in$  acceptor :  $maxBal(A) = -1$ 
13 Next  $\triangleq \exists A \in$  acceptor,  $B \in$  ballot,  $V \in$  value :
   IncreaseMaxBal( $A, B$ )  $\vee$  VoteFor( $A, B, V$ )
14 Safety  $\triangleq \forall V_1, V_2 \in$  value : chosen( $V_1$ )  $\wedge$  chosen( $V_2$ )  $\rightarrow V_1 = V_2$ 

```

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) :

```

value2  $\triangleq$  {v1, v2}
acceptor3  $\triangleq$  {a1, a2, a3}
quorum3  $\triangleq$  {q12: {a1, a2}, q13: {a1, a3}, q23: {a2, a3}}
ballot4  $\triangleq$  [bmin, b1, b2, bmax]

```

has three finite symmetric sorts named value₂, acceptor₃ and quorum₃, defined as sets of arbitrarily-named distinct constants, while the finite totally-ordered sort ballot₄ is composed of a list of ordered constants, i.e., $b_{\min} < b_1 < b_2 < b_{\max}$, where $b_{\min} = -1$ since -1 is the “minimum” ballot number. The constants of the quorum₃ sort are subsets of the acceptor₃ sort and are named to reflect

```

1 CONSTANTS value, acceptor, quorum
2 ballot  $\triangleq$  Nat  $\cup$   $\{-1\}$ 
3 VARIABLES msg1a, msg1b, msg2a, msg2b, maxBal
   maxVBal, maxVal
4 msg1a  $\in$  ballot  $\rightarrow$  BOOLEAN
   msg1b  $\in$  (acceptor  $\times$  ballot  $\times$  ballot  $\times$  value)  $\rightarrow$  BOOLEAN
   msg2a  $\in$  (ballot  $\times$  value)  $\rightarrow$  BOOLEAN
   msg2b  $\in$  (acceptor  $\times$  ballot  $\times$  value)  $\rightarrow$  BOOLEAN
   maxBal  $\in$  acceptor  $\rightarrow$  ballot
   maxVBal  $\in$  acceptor  $\rightarrow$  ballot
   maxVal  $\in$  acceptor  $\rightarrow$  value
   none  $\in$  value
5 ASSUME  $\wedge \forall Q \in$  quorum :  $Q \subseteq$  acceptor
    $\wedge \forall Q_1, Q_2 \in$  quorum :  $Q_1 \cap Q_2 \neq \{\}$ 
6 chosenAt( $b, v$ )  $\triangleq \exists Q \in$  quorum :  $\forall A \in Q : msg2b(A, b, v)$ 
7 chosen( $v$ )  $\triangleq \exists B \in$  ballot : chosenAt( $B, v$ )
8 showsSafeAtPaxos( $q, b, v$ )  $\triangleq$ 
    $\wedge \forall A \in q : \exists M_b \in$  ballot :  $\exists M_v \in$  value : msg1b( $A, b, M_b, M_v$ )
    $\wedge \forall A \in$  acceptor :  $\forall M_b \in$  ballot :  $\forall M_v \in$  value :
      $\neg (A \in q \wedge msg1b(A, b, M_b, M_v) \wedge (M_b \neq -1))$ 
      $\vee \exists M_b \in$  ballot :
        $\wedge \exists A \in q : msg1b(A, b, M_b, v) \wedge (M_b \neq -1)$ 
        $\wedge \forall A \in q : \forall M_{b2} \in$  ballot :  $\forall M_{v2} \in$  value :
         msg1b( $A, b, M_{b2}, M_{v2}$ )  $\wedge (M_{b2} \neq -1) \rightarrow M_{b2} \leq M_b$ 
9 isSafeAtPaxos( $b, v$ )  $\triangleq \exists Q \in$  quorum : showsSafeAtPaxos( $Q, b, v$ )
10 Phase1a( $b$ )  $\triangleq$ 
    $\wedge b \neq -1$ 
    $\wedge msg1a' = [msg1a \text{ EXCEPT } ![b] = \top]$ 
    $\wedge$  UNCHANGED msg1b, msg2a, msg2b, maxBal, maxVBal, maxVal
11 Phase1b( $a, b$ )  $\triangleq$ 
    $\wedge b \neq -1 \wedge msg1a(b) \wedge b > maxBal(a)$ 
    $\wedge maxBal' = [maxBal \text{ EXCEPT } ![a] = b]$ 
    $\wedge msg1b' = [msg1b \text{ EXCEPT } ![a, b, maxVBal(a), maxVal(a)] = \top]$ 
    $\wedge$  UNCHANGED msg1a, msg2a, msg2b, maxVBal, maxVal
12 Phase2a( $b, v$ )  $\triangleq$ 
    $\wedge b \neq -1 \wedge v \neq none \wedge \neg (\exists V \in$  value : msg2a( $b, V$ ))
    $\wedge isSafeAtPaxos(b, v)$ 
    $\wedge msg2a' = [msg2a \text{ EXCEPT } ![b, v] = \top]$ 
    $\wedge$  UNCHANGED msg1a, msg1b, msg2b, maxBal, maxVBal, maxVal
13 Phase2b( $a, b, v$ )  $\triangleq$ 
    $\wedge b \neq -1 \wedge v \neq none \wedge msg2a(b, v) \wedge b \geq maxBal(a)$ 
    $\wedge maxBal' = [maxBal \text{ EXCEPT } ![a] = b]$ 
    $\wedge maxVBal' = [maxVBal \text{ EXCEPT } ![a] = b]$ 
    $\wedge maxVal' = [maxVal \text{ EXCEPT } ![a] = v]$ 
    $\wedge msg2b' = [msg2b \text{ EXCEPT } ![a, b, v] = \top]$ 
    $\wedge$  UNCHANGED msg1a, msg1b, msg2a
14 Init  $\triangleq \forall A \in$  acceptor :  $B \in$  ballot :
    $\wedge \neg msg1a(B)$ 
    $\wedge \forall M_b \in$  ballot :  $M_v \in$  value :  $\neg msg1b(A, B, M_b, M_v)$ 
    $\wedge \forall V \in$  value :  $\neg msg2a(B, V) \wedge \neg msg2b(A, B, V)$ 
    $\wedge maxBal(A) = -1$ 
    $\wedge maxVBal(A) = -1 \wedge maxVal(A) = none$ 
15 Next  $\triangleq \exists A \in$  acceptor :  $B \in$  ballot :  $V \in$  value :
    $\vee$  Phase1a( $B$ )  $\vee$  Phase1b( $A, B$ )
    $\vee$  Phase2a( $B, V$ )  $\vee$  Phase2b( $A, B, V$ )
16 Safety  $\triangleq \forall V_1, V_2 \in$  value : chosen( $V_1$ )  $\wedge$  chosen( $V_2$ )  $\rightarrow V_1 = V_2$ 

```

Fig. 3: Lamport's *Paxos* protocol in pretty-printed TLA+

their symmetric dependence on the acceptor_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_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 $[votes \leftarrow \text{msg2b}, \text{maxBal} \leftarrow \text{maxBal}]$, where acceptors now communicate with each other through distributed message passing. State variables msg1a , msg1b , msg2a , and msg2b are used to model the set of different messages that can be sent in the protocol, corresponding to actions *Phase1a*, *Phase1b*, *Phase2a*, and *Phase2b* respectively. The pair $\langle \text{maxVBal}(a), \text{maxVal}(a) \rangle$ is the vote with the largest ballot number cast by acceptor a . The ballot b leader can send a $\text{msg1a}(b)$ by performing the action *Phase1a*(b). *Phase1b*(a, b) implements the *IncreaseMaxBal*(a, b) action from *Voting*, where after receiving $\text{msg1a}(b)$, acceptor a sends msg1b to the ballot b leader containing the values of $\text{maxVBal}(a)$ and $\text{maxVal}(a)$. In the *Phase2a*(b, v) action, the ballot b leader sends msg2a asking the acceptors to vote for a value v that is safe at ballot number b . Its enabling condition $\text{isSafeAtPaxos}(b, v)$ checks the enabling condition $\text{isSafeAt}(b, v)$ from *Voting*. *Phase2b* 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 $\text{Paxos}(\text{value}, \text{acceptor}, \text{quorum}, \text{ballot})$, its finite instance $\text{Paxos}(2, 3, 3, 4)$ has 132 BOOLEAN state variables, 6 state variables that return a ballot number in ballot_4 , and 3 state variables that return a value in value_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 $\text{maxVBal}(a)$ and $\text{maxVal}(a)$, *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 msg2b from acceptor a , by modifying the *Phase1b*(a, b) action (line 11 in Figure 3) to 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 msg1b completely, i.e., the last two arguments of msg1b 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

```

MODULE ImplicitPaxos
11 Phase1b(a, b)  $\triangleq$ 
 $\wedge b \neq -1 \wedge \text{msg1a}(b) \wedge b > \text{maxBal}(a)$ 
 $\wedge \text{maxBal}' = [\text{maxBal} \text{ EXCEPT } ![a] = b]$ 
 $\wedge \exists M_b \in \text{ballot} : \exists M_v \in \text{value} :$ 
 $\wedge \vee \wedge (M_b = -1)$ 
 $\wedge \forall B \in \text{ballot} : \forall V \in \text{value} : \neg \text{msg2b}(a, B, V)$ 
 $\vee \wedge (M_b \neq -1) \wedge \text{msg2b}(a, M_b, M_v)$ 
 $\wedge \forall B \in \text{ballot} : \forall V \in \text{value} :$ 
 $\text{msb2b}(a, B, V) \rightarrow B \leq M_b$ 
 $\wedge \text{msg1b}' = [\text{msg1b} \text{ EXCEPT } ![a, M_b, M_v] = \top]$ 
 $\wedge \text{UNCHANGED } \text{msg1a}, \text{msg2a}, \text{msg2b}$ 

```

Fig. 4: Modifications in *ImplicitPaxos* compared to *Paxos*

```

MODULE SimplePaxos
8 showsSafeAtSimplePaxos(q, b, v)  $\triangleq$ 
 $\wedge \forall A \in q : \text{msg1b}(A, b)$ 
 $\wedge \forall A \in \text{acceptor} : \forall M_b \in \text{ballot} : \forall M_v \in \text{value} :$ 
 $\neg (A \in q \wedge \text{msg1b}(A, b) \wedge \text{msg2b}(A, M_b, M_v))$ 
 $\vee \exists M_b \in \text{ballot} :$ 
 $\wedge \exists A \in q : \text{msg1b}(A, b) \wedge \text{msg2b}(A, M_b, v)$ 
 $\wedge \forall A \in q : \forall M_{b2} \in \text{ballot} : \forall M_{v2} \in \text{value} :$ 
 $\text{msg1b}(A, b) \wedge \text{msg2b}(A, M_{b2}, M_{v2}) \rightarrow M_{b2} \leq M_b$ 

```

Fig. 5: Modifications in *SimplePaxos* compared to *ImplicitPaxos*

showsSafeAtPaxos (line 8 in Figure 3) with its simplified form, expressed using msg2b as shown in Figure 5.

VI. HIERARCHICAL VERIFICATION OF PAXOS

Using the 4-level hierarchy $\text{Paxos} \prec \text{ImplicitPaxos} \prec \text{SimplePaxos} \prec \text{Voting}$, 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 $\text{Voting}(2, 3, 3, 4)$, IC3PO proved the safety of *Voting* by automatically deriving the inductive invariant $V!inv \triangleq V!Safety \wedge V!A_1 \wedge V!A_2$ 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:

- A_1 : If an acceptor voted for value V in ballot number B , then V is safe at B .
- 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 .

B. Proving SimplePaxos

Using the refinement mapping $[votes \leftarrow \text{msg2b}, \text{maxBal} \leftarrow \text{maxBal}]$, IC3PO transformed $V!A_1$ and $V!A_2$ to the follow-

ing corresponding versions for *SimplePaxos*:

$$\begin{aligned} S!A_1 &= \forall A \in \text{acceptor}, B \in \text{ballot}, V \in \text{value} : \\ &\quad \text{msg2b}(A, B, V) \rightarrow \text{isSafeAt}(B, V) \\ S!A_2 &= \forall A \in \text{acceptor}, B \in \text{ballot}, V_1, V_2 \in \text{value} : \\ &\quad \text{chosenAt}(B, V_1) \wedge \text{msg2b}(A, B, V_2) \rightarrow (V_1 = V_2) \end{aligned}$$

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 $S!inv \triangleq S!Safety \wedge \bigwedge_{1 \leq i \leq 6} S!A_i$ where

$$\begin{aligned} S!A_3 &= \forall B \in \text{ballot}, V \in \text{value} : \\ &\quad \text{msg2a}(B, V) \rightarrow \text{isSafeAt}(B, V) \\ S!A_4 &= \forall B \in \text{ballot}, V_1, V_2 \in \text{value} : \\ &\quad \text{msg2a}(B, V_1) \wedge \text{msg2a}(B, V_2) \rightarrow (V_1 = V_2) \\ S!A_5 &= \forall A \in \text{acceptor}, B \in \text{ballot}, V \in \text{value} : \\ &\quad \text{msg2b}(A, B, V) \rightarrow \text{msg2a}(B, V) \\ S!A_6 &= \forall A \in \text{acceptor}, B \in \text{ballot} : \\ &\quad \text{msg1b}(A, B) \rightarrow \text{maxBal}(A) \geq B \end{aligned}$$

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

- A_3 : If ballot B leader sends a $2a$ message for value V , then V is safe at B .
- A_4 : A ballot leader can send $2a$ messages only for a unique value.
- A_5 : If an acceptor voted for a value in ballot number B , then there is a $2a$ message for that value at B .
- A_6 : 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 $I!A_1$ to $I!A_5$ in *ImplicitPaxos* as is, while $S!A_6$ maps as:

$$I!A_6 = \forall A \in \text{acceptor}, B, B_{max} \in \text{ballot}, V_{max} \in \text{value} : \\ \text{msg1b}(A, B, B_{max}, V_{max}) \rightarrow \text{maxBal}(A) \geq B$$

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 $I!inv \triangleq I!Safety \wedge \bigwedge_{1 \leq i \leq 8} I!A_i$ where

$$\begin{aligned} I!A_7 &= \forall A \in \text{acceptor}, B, B_{max} \in \text{ballot}, V_{max} \in \text{value} : \\ &\quad [(B > -1) \wedge (B_{max} > -1) \wedge \text{msg1b}(A, B, B_{max}, V_{max})] \\ &\quad \rightarrow \text{msg2b}(A, B_{max}, V_{max}) \\ I!A_8 &= \forall A \in \text{acceptor}, B, B_{mid}, B_{max} \in \text{ballot}, \\ &\quad V, V_{max} \in \text{value} : \\ &\quad [(B > B_{mid}) \wedge (B_{mid} > B_{max}) \wedge \text{msg1b}(A, B, B_{max}, V_{max})] \\ &\quad \rightarrow \neg \text{msg2b}(A, B_{mid}, V) \end{aligned}$$

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

- A_7 : 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} .
- A_8 : 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 $I!A_1, \dots, I!A_8$ to $P!A_1, \dots, P!A_6$ 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 $P!inv \triangleq P!Safety \wedge \bigwedge_{1 \leq i \leq 11} P!A_i$ where

$$\begin{aligned} P!A_9 &= \forall A \in \text{acceptor} : \text{maxVBal}(A) \leq \text{maxBal}(A) \\ P!A_{10} &= \forall A \in \text{acceptor}, B \in \text{ballot}, V \in \text{value} : \\ &\quad \text{msg2b}(A, B, V) \rightarrow \text{maxVBal}(A) \geq B \\ P!A_{11} &= \forall A \in \text{acceptor} : \text{maxVBal}(A) > -1 \\ &\quad \rightarrow \text{msg2b}(A, \text{maxVBal}(A), \text{maxVal}(A)) \end{aligned}$$

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

- A_9 : maxVBal of an acceptor is less than or equal to its maxBal .
- A_{10} : If an acceptor voted in a ballot number B , then its maxVBal is at least as high as B .
- A_{11} : If acceptor A has its 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 $P!inv$ 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 $A_3 \wedge A_4 \wedge A_5$. 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., $V!inv$, 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 *infinite* totally-ordered 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 *isSafeAt* and *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 *Paxos* \prec *Voting*. Given the two strengthening assertions A_1 and A_2 from *Voting*, inferring the remaining nine assertions for *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 *Paxos* and learnt too many weak clauses involving *msg1b*, *maxVBal* and *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], *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 *ImplicitPaxos* reduced the complexity in *Paxos* by abstracting away *maxVBal* and *maxVal*. Still, scalability remained a challenge due to *msg1b*, that contributed to 96 out of 147 state bits in *Paxos*(2, 3, 3, 4). Adding another level, i.e., *SimplePaxos*, removed 84 out of these 96 state bits by abstracting away explicit tracking of the maximum vote of

Finite Instance	State-space Size
<i>Voting</i> (2, 3, 3, 4)	2^{30}
<i>SimplePaxos</i> (2, 3, 3, 4)	2^{54}
<i>ImplicitPaxos</i> (2, 3, 3, 4)	2^{138}
<i>Paxos</i> (2, 3, 3, 4)	2^{147}

TABLE I: State-space size for finite instances with 2 value, 3 acceptor, 3 quorum, and 4 ballot

an acceptor from *msg1b*. When compared against *Paxos*, *SimplePaxos* is significantly simpler, with a total state-space size to be just 2^{54} for its finite instance *SimplePaxos*(2, 3, 3, 4), which led IC3PO to successfully prove *Paxos* automatically using the four-level hierarchy.

F. Extension to MultiPaxos and FlexiblePaxos

Till now, by *Paxos* we meant *single-decree Paxos* which is the core consensus algorithm underlying the complete Paxos state-machine replication protocol [1], [2], commonly referred to as *MultiPaxos* [43]. In *MultiPaxos*, a sequence of instances execute *single-decree Paxos* such that the value chosen in the i^{th} instance becomes the i^{th} command executed by the replicated state machine. Additionally, if the leader is relatively stable, *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, \dots, A_{11} to *MultiPaxos* is trivial, and simply adds the corresponding instance as an additional universally-quantified argument, e.g., A_{11} maps as:

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

Unsurprisingly, the 11 strengthening assertions, passed down from the proof of *Paxos*, together with the safety property of *MultiPaxos*, allowed IC3PO to trivially prove it with no additional strengthening assertions needed, meaning $M!Safety \wedge \bigwedge_{1 \leq i \leq 11} M!A_i$ is already an inductive invariant of *MultiPaxos*. As described in previous works [1], [2], [6], [10], the crux of proving the safety of *MultiPaxos* is based on proving *single-decree 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 *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 *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 *mypyvy* [58], extends IC3 with a separators-based technique that performs enumer-

Protocol	S.A.	Time (seconds)						Inv		SMT		
		IC3PO	SWISS	fol-ic3	DistAI	I4	UPDR	IC3PO	Human	IC3PO	I4	
EPR	epr-paxos	\emptyset	568	15950*	timeout	error	memout	timeout	6	11	5680	1701556
	epr-flexible_paxos	\emptyset	561	18232*	timeout	error	memout	failure	6	11	1509	1761504
	epr-multi_paxos	\emptyset	timeout	timeout	timeout	error	memout	timeout	—	12	—	1902621
ORIGINAL	<i>Voting</i>	\emptyset	64	timeout	timeout	error	memout	timeout	3	3	1057	1714170
	<i>SimplePaxos</i>	A_{1-2}	51	timeout	timeout	error	failure	timeout	5	5	618	158470
	<i>ImplicitPaxos</i>	A_{1-6}	2008	timeout	timeout	error	failure	timeout	7	7	18329	69715
	<i>Paxos</i>	A_{1-8}	98	timeout	timeout	error	failure	timeout	10	10	668	76030
	<i>MultiPaxos</i>	A_{1-11}	340	timeout	timeout	error	timeout	timeout	10	10	161	—
	<i>FlexiblePaxos</i>	A_{1-11}	1408	timeout	timeout	error	failure	timeout	10	10	161	6983

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 (\emptyset 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).

ative search for a quantified separator in the space of bounded mixed quantifier prefixes.

- DistAI [50] performs data-driven invariant learning by enumerating over possible invariants derived from simulating a protocol at different instance sizes, followed by iteratively refining and checking candidate invariants.
- I4 [51], [59] performs finite-domain IC3 (without accounting for regularity) using the AVR model checker [55], [60], followed by iteratively generalizing and checking the inductive invariant produced by AVR.
- UPDR, from the *mypyvy* [58] framework, implements PDR[∇]/UPDR [61] for verifying distributed protocols.

All experiments were performed on an Intel (R) Xeon CPU (X5670). For each run, we used a 5-hour timeout and a 32 GB memory limit. All tools were executed in their respective default configurations. We used Z3 [62] version 4.8.10, Yices 2 [63] version 2.6.2, and CVC4 [64] version 1.8.

A. Results

Table II summarizes the experimental results. EPR variants were run without any hierarchical strengthening. For ORIGINAL problems, we employed hierarchical strengthening using each tool to verify Lamport’s original Paxos specification (and its variants) through higher-level strengthening assertions that were automatically generated from IC3PO (as detailed in Section VI). Note that ORIGINAL problems include quantifier-alternation cycles that make unbounded SMT reasoning fall into the undecidable fragment of first-order logic.

IC3PO emerges as the only successful technique that verifies Lamport’s Paxos and its variants, and automatically infers the required inductive invariants efficiently. Unsurprisingly, none of the other tools (i.e., SWISS, fol-ic3, DistAI, I4 and UPDR) were able to solve ORIGINAL problems since each of these tools rely on unbounded SMT reasoning and struggle on problems that fall outside the decidable EPR fragment of first-order logic.

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 A_1 to A_{11} 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/aman-goel/fmcd2021exp>.

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

- [1] L. Lamport, “The part-time parliament,” *ACM Trans. Comput. Syst.*, vol. 16, no. 2, p. 133–169, May 1998. [Online]. Available: <https://doi.org/10.1145/279227.279229>
- [2] —, “Paxos made simple,” pp. 51–58, December 2001. [Online]. Available: <https://www.microsoft.com/en-us/research/publication/paxos-made-simple/>
- [3] M. Burrows, “The chubby lock service for loosely-coupled distributed systems,” in *Proceedings of the 7th symposium on Operating systems design and implementation*, 2006, pp. 335–350.
- [4] T. D. Chandra, R. Griesemer, and J. Redstone, “Paxos made live: An engineering perspective,” in *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC ’07. New York, NY, USA: Association for Computing Machinery, 2007, p. 398–407. [Online]. Available: <https://doi.org/10.1145/1281100.1281103>
- [5] M. Isard, “Autopilot: Automatic data center management,” *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 2, p. 60–67, Apr. 2007. [Online]. Available: <https://doi.org/10.1145/1243418.1243426>
- [6] R. De Prisco, B. Lampson, and N. Lynch, “Revisiting the paxos algorithm,” *Theoretical Computer Science*, vol. 243, no. 1-2, pp. 35–91, 2000.
- [7] K. Chaudhuri, D. Doligez, L. Lamport, and S. Merz, “The tla+ proof system: Building a heterogeneous verification platform,” in *International Colloquium on Theoretical Aspects of Computing*. Springer, 2010, pp. 44–44.
- [8] D. Cousineau, D. Doligez, L. Lamport, S. Merz, D. Ricketts, and H. Vanzetto, “Tla+ proofs,” in *FM 2012: Formal Methods*, D. Gianakopoulou and D. Méry, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 147–154.
- [9] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL: a proof assistant for higher-order logic*. Springer Science & Business Media, 2002, vol. 2283.

- [10] S. Chand, Y. A. Liu, and S. D. Stoller, “Formal verification of multi-paxos for distributed consensus,” in *International Symposium on Formal Methods*. Springer, 2016, pp. 119–136.
- [11] L. Lamport, *Specifying Systems*. Addison-Wesley Boston, 2002, vol. 388.
- [12] O. Padon, G. Losa, M. Sagiv, and S. Shoham, “Paxos made epr: decidable reasoning about distributed protocols,” *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, pp. 108:1–108:31, 2017.
- [13] O. Padon, K. L. McMillan, A. Panda, M. Sagiv, and S. Shoham, “Ivy: safety verification by interactive generalization,” in *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2016, pp. 614–630.
- [14] R. Piskac, L. de Moura, and N. Bjørner, “Deciding effectively propositional logic using dpll and substitution sets,” *Journal of Automated Reasoning*, vol. 44, no. 4, pp. 401–424, 2010.
- [15] L. Lamport, “Byzantizing paxos by refinement,” in *International Symposium on Distributed Computing*. Springer, 2011, pp. 211–224.
- [16] C. Hawblitzel, J. Howell, M. Kapritsos, J. R. Lorch, B. Parno, M. L. Roberts, S. Setty, and B. Zill, “Ironfleet: proving practical distributed systems correct,” in *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM, 2015, pp. 1–17.
- [17] J. R. Wilcox, D. Woos, P. Panchekha, Z. Tatlock, X. Wang, M. D. Ernst, and T. Anderson, “Verdi: A framework for implementing and formally verifying distributed systems,” in *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI ’15. New York, NY, USA: ACM, 2015, pp. 357–368. [Online]. Available: <http://doi.acm.org/10.1145/2737924.2737958>
- [18] S. Merz, “Formal specification and verification,” in *Concurrency: the Works of Leslie Lamport*, 2019, pp. 103–129.
- [19] B. Kragl, S. Qadeer, and T. A. Henzinger, “Refinement for structured concurrent programs,” in *International Conference on Computer Aided Verification*. Springer, 2020, pp. 275–298.
- [20] M. Abadi and L. Lamport, “The existence of refinement mappings,” *Theoretical Computer Science*, vol. 82, no. 2, pp. 253–284, 1991.
- [21] L. Lamport, “The temporal logic of actions,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 16, no. 3, pp. 872–923, 1994.
- [22] —, “Refinement in state-based formalisms,” *Digital Equipment Corporation*, 1996.
- [23] S. J. Garland and N. A. Lynch, “Using i/o automata for developing distributed systems,” *Foundations of component-based systems*, vol. 13, no. 285-312, pp. 5–2, 2000.
- [24] A. R. Bradley, “SAT-Based Model Checking without Unrolling,” in *Proceedings of the 12th international conference on Verification, model checking, and abstract interpretation*, ser. VMCAI’11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 70–87. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1946284.1946291>
- [25] N. Een, A. Mishchenko, and R. Brayton, “Efficient Implementation of Property Directed Reachability,” in *Formal Methods in Computer Aided Design (FMCAD’11)*, Oct. 2011, pp. 125 – 134.
- [26] E. M. Clarke, E. A. Emerson, and J. Sifakis, “Model checking: algorithmic verification and debugging,” *Communications of the ACM*, vol. 52, no. 11, pp. 74–84, 2009.
- [27] A. Goel and K. Sakallah, “On symmetry and quantification: A new approach to verify distributed protocols,” in *NASA Formal Methods*, A. Dutle, M. M. Moscato, L. Titolo, C. A. Muñoz, and I. Perez, Eds. Cham: Springer International Publishing, 2021, pp. 131–150. [Online]. Available: https://doi.org/10.1007/978-3-030-76384-8_9
- [28] D. Doligez, L. Lamport, and S. Merz, “A TLA+ specification of the Paxos consensus algorithm and a TLAPS-checked proof of its correctness,” <https://github.com/tlaplus/tlapm/blob/master/examples/paxos/Paxos.tla>.
- [29] M. Taube, G. Losa, K. L. McMillan, O. Padon, M. Sagiv, S. Shoham, J. R. Wilcox, and D. Woos, “Modularity for decidability of deductive verification with applications to distributed systems,” in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2018, pp. 662–677.
- [30] C. Barrett, P. Fontaine, and C. Tinelli, “The Satisfiability Modulo Theories Library (SMT-LIB),” www.SMT-LIB.org, 2016.
- [31] A. Goel and K. A. Sakallah, “On symmetry and quantification: A new approach to verify distributed protocols,” *CoRR*, vol. abs/2103.14831, 2021. [Online]. Available: <https://arxiv.org/abs/2103.14831>
- [32] A. Pnueli, S. Ruah, and L. Zuck, “Automatic deductive verification with invisible invariants,” in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2001, pp. 82–97.
- [33] T. Arons, A. Pnueli, S. Ruah, Y. Xu, and L. Zuck, “Parameterized verification with automatically computed inductive assertions,” in *Computer Aided Verification*, G. Berry, H. Comon, and A. Finkel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 221–234.
- [34] L. Zuck and A. Pnueli, “Model checking and abstraction to the aid of parameterized systems (a survey),” *Computer Languages, Systems & Structures*, vol. 30, no. 3-4, pp. 139–169, 2004.
- [35] I. Balaban, Y. Fang, A. Pnueli, and L. D. Zuck, “Iiv: An invisible invariant verifier,” in *International Conference on Computer Aided Verification*. Springer, 2005, pp. 408–412.
- [36] K. S. Namjoshi, “Symmetry and completeness in the analysis of parameterized systems,” in *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, 2007, pp. 299–313.
- [37] L. Lamport, “How to write a proof,” *The American mathematical monthly*, vol. 102, no. 7, pp. 600–608, 1995.
- [38] —, “A TLA+ specification of the Voting algorithm from Leslie Lamport’s lectures titled: The Paxos Algorithm - or How to Win a Turing Award.” <https://github.com/tlaplus/Examples/blob/master/specifications/PaxosHowToWinATuringAward/Voting.tla>, 2019.
- [39] —, “A TLA+ specification of the Paxos Consensus algorithm from Leslie Lamport’s lectures titled: The Paxos Algorithm - or How to Win a Turing Award.” <https://github.com/tlaplus/Examples/blob/master/specifications/PaxosHowToWinATuringAward/Paxos.tla>, 2019.
- [40] —, “The Paxos Algorithm - or How to Win a Turing Award.” <https://lamport.azurewebsites.net/tla/paxos-algorithm.html?back-link=more-stuff.html>, 2019.
- [41] —, “Generalized consensus and paxos,” Tech. Rep. MSR-TR-2005-33, March 2005. [Online]. Available: <https://www.microsoft.com/en-us/research/publication/generalized-consensus-and-paxos/>
- [42] S. Peluso, A. Turcu, R. Palmieri, G. Losa, and B. Ravindran, “Making fast consensus generally faster,” in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016, pp. 156–167.
- [43] “A TLA+ specification of the MultiPaxos algorithm.” <https://github.com/tlaplus/Examples/tree/master/specifications/MultiPaxos>.
- [44] G. Losa, “Paxos consensus protocol in Ivy.” <https://github.com/nano-0/ivy-proofs/blob/master/paxos/paxos.ivy>.
- [45] L. Lamport and S. Merz, “A TLA+ specification of the Voting algorithm and a TLAPS-checked proof of its correctness,” <https://github.com/tlaplus/tlapm/blob/master/examples/ByzPaxos/VoteProof.tla>.
- [46] L. Lamport, “Proving the correctness of multiprocess programs,” *IEEE transactions on software engineering*, no. 2, pp. 125–143, 1977.
- [47] S. Owicki and D. Gries, “Verifying properties of parallel programs: An axiomatic approach,” *Communications of the ACM*, vol. 19, no. 5, pp. 279–285, 1976.
- [48] T. Hance, M. Heule, R. Martins, and B. Parno, “Finding invariants of distributed systems: It’s a small (enough) world after all,” in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021, pp. 115–131. [Online]. Available: <https://www.usenix.org/conference/nsdi21/presentation/hance>
- [49] J. R. Koenig, O. Padon, N. Immerman, and A. Aiken, “First-order quantified separators,” in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 703–717. [Online]. Available: <https://github.com/wilcoxjay/myppyvy/tree/pldi20-artifact>
- [50] J. Yao, R. Tao, R. Gu, J. Nieh, S. Jana, and G. Ryan, “Distai: Data-driven automated invariant learning for distributed protocols,” in *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, 2021, pp. 405–421.
- [51] H. Ma, A. Goel, J.-B. Jeannin, M. Kapritsos, B. Kasikci, and K. A. Sakallah, “I4: Incremental inference of inductive invariants for verification of distributed protocols,” in *Proceedings of the 27th Symposium on Operating Systems Principles*. ACM, 2019.
- [52] A. Karbyshev, N. Bjørner, S. Itzhaky, N. Rinetzky, and S. Shoham, “Property-directed inference of universal invariants or proving their absence,” *Journal of the ACM (JACM)*, vol. 64, no. 1, pp. 1–33, 2017. [Online]. Available: <https://bitbucket.org/tausigplan/updr-distrib/src/master/>

- [53] A. Biere, N. Froyeys, and M. Preiner, “Hardware model checking competition (HWMCC) 2020,” <http://fmv.jku.at/hwmcc20>.
- [54] A. Goel and K. Sakallah, “Empirical evaluation of ic3-based model checking techniques on verilog rtl designs,” in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2019, pp. 618–621.
- [55] A. Goel and K. Sakallah, “Model checking of verilog rtl using ic3 with syntax-guided abstraction,” in *NASA Formal Methods*, J. M. Badger and K. Y. Rozier, Eds. Cham: Springer International Publishing, 2019, pp. 166–185.
- [56] H. Howard, D. Malkhi, and A. Spiegelman, “Flexible paxos: Quorum intersection revisited,” *CoRR*, vol. abs/1608.06696, 2016. [Online]. Available: <http://arxiv.org/abs/1608.06696>
- [57] A. Goel and K. A. Sakallah, “IC3PO: IC3 for Proving Protocol Properties,” <https://github.com/aman-goel/ic3po>.
- [58] “mypyvy on GitHub,” <https://github.com/wilcoxjay/mypyvy>.
- [59] H. Ma, A. Goel, J.-B. Jeannin, M. Kapritsos, B. Kasikci, and K. A. Sakallah, “Towards automatic inference of inductive invariants,” in *Proceedings of the Workshop on Hot Topics in Operating Systems*. ACM, 2019, pp. 30–36.
- [60] A. Goel and K. Sakallah, “AVR: Abstractly Verifying Reachability,” <http://www.github.com/aman-goel/avr>.
- [61] A. Karbyshev, N. Bjørner, S. Itzhaky, N. Rinetsyky, and S. Shoham, “Property-directed inference of universal invariants or proving their absence,” *J. ACM*, vol. 64, no. 1, Mar. 2017. [Online]. Available: <https://doi.org/10.1145/3022187>
- [62] L. De Moura and N. Bjørner, “Z3: An efficient SMT solver,” in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.
- [63] B. Dutertre, “Yices 2.2,” in *Computer Aided Verification*, A. Biere and R. Bloem, Eds. Cham: Springer International Publishing, 2014, pp. 737–744.
- [64] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli, “CVC4,” in *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV ’11)*, ser. Lecture Notes in Computer Science, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, Jul. 2011, pp. 171–177. snowbird, Utah. [Online]. Available: <http://www.cs.stanford.edu/~barrett/pubs/BCD+11.pdf>
- [65] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, and M. Dearduff, “How amazon web services uses formal methods,” *Communications of the ACM*, vol. 58, no. 4, pp. 66–73, 2015.
- [66] R. Beers, “Pre-RTL formal verification: an intel experience,” in *Proceedings of the 45th annual Design Automation Conference*, 2008, pp. 806–811.
- [67] “The TLA+ Toolbox,” <https://lambort.azurewebsites.net/tla-toolbox.html>.
- [68] Y. Yu, P. Manolios, and L. Lamport, “Model checking tla+ specifications,” in *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*. Springer, 1999, pp. 54–66.
- [69] K. Chaudhuri, D. Doligez, L. Lamport, and S. Merz, “Verifying safety properties with the tla+ proof system,” in *International Joint Conference on Automated Reasoning*. Springer, 2010, pp. 142–148.
- [70] C. Drăgoi, T. A. Henzinger, and D. Zufferey, “Psync: a partially synchronous language for fault-tolerant distributed algorithms,” *ACM SIGPLAN Notices*, vol. 51, no. 1, pp. 400–415, 2016.
- [71] J. Hoenicke, R. Majumdar, and A. Podelski, “Thread modularity at many levels: a pearl in compositional verification,” *ACM SIGPLAN Notices*, vol. 52, no. 1, pp. 473–485, 2017.
- [72] K. v. Gleissenthall, R. G. Kıcı, A. Bakst, D. Stefan, and R. Jhala, “Pre-tend synchrony: synchronous verification of asynchronous distributed programs,” *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–30, 2019.
- [73] C. N. Ip and D. L. Dill, “Better verification through symmetry,” in *Computer Hardware Description Languages and their Applications*. Elsevier, 1993, pp. 97–111.
- [74] C. Norris IP and D. L. Dill, “Better verification through symmetry,” *Formal Methods in System Design*, vol. 9, no. 1, pp. 41–75, Aug 1996. [Online]. Available: <https://doi.org/10.1007/BF00625968>
- [75] E. M. Clarke, T. Filkorn, and S. Jha, “Exploiting symmetry in temporal logic model checking,” in *International Conference on Computer Aided Verification*. Springer, 1993, pp. 450–462.
- [76] E. A. Emerson and K. S. Namjoshi, “Reasoning about rings,” in *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1995, pp. 85–94.
- [77] E. A. Emerson and A. P. Sistla, “Symmetry and model checking,” *Formal methods in system design*, vol. 9, no. 1-2, pp. 105–131, 1996.
- [78] A. P. Sistla, V. Gyuris, and E. A. Emerson, “Smc: a symmetry-based model checker for verification of safety and liveness properties,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 9, no. 2, pp. 133–166, 2000.
- [79] K. R. Apt and D. Kozen, “Limits for automatic verification of finite-state concurrent systems,” *Inf. Process. Lett.*, vol. 22, no. 6, pp. 307–309, 1986.
- [80] A. Gurfinkel, S. Shoham, and Y. Vizel, “Quantifiers on demand,” in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2018, pp. 248–266.
- [81] Y. M. Feldman, J. R. Wilcox, S. Shoham, and M. Sagiv, “Inferring inductive invariants from phase structures,” in *International Conference on Computer Aided Verification*. Springer, 2019, pp. 405–425.
- [82] J. R. Koenig, O. Padon, N. Immerman, and A. Aiken, “First-order quantified separators,” in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 703–717. [Online]. Available: <https://doi.org/10.1145/3385412.3386018>
- [83] S. Ranise and S. Ghilardi, “Backward reachability of array-based systems by smt solving: Termination and invariant synthesis,” *Logical Methods in Computer Science*, vol. 6, 2010.
- [84] S. Conchon, A. Goel, S. Krstić, A. Mebsout, and F. Zaïdi, “Cubicle: A parallel smt-based model checker for parameterized systems,” in *International Conference on Computer Aided Verification*. Springer, 2012, pp. 718–724.
- [85] Y. Li, J. Pang, Y. Lv, D. Fan, S. Cao, and K. Duan, “Paraverifier: An automatic framework for proving parameterized cache coherence protocols,” in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2015, pp. 207–213.
- [86] P. Abdulla, F. Haziza, and L. Holík, “Parameterized verification through view abstraction,” *International Journal on Software Tools for Technology Transfer*, vol. 18, no. 5, pp. 495–516, 2016.
- [87] M. Dooley and F. Somenzi, “Proving parameterized systems safe by generalizing clausal proofs of small instances,” in *International Conference on Computer Aided Verification*. Springer, 2016, pp. 292–309.
- [88] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, “Counterexample-Guided Abstraction Refinement,” in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, E. Emerson and A. Sistla, Eds. Springer Berlin / Heidelberg, 2000, vol. 1855, pp. 154–169, 10.1007/10722167_15. [Online]. Available: http://dx.doi.org/10.1007/10722167_15
- [89] —, “Counterexample-Guided Abstraction Refinement for Symbolic Model Checking,” *J. ACM*, vol. 50, pp. 752–794, September 2003. [Online]. Available: <http://doi.acm.org.proxy.lib.umich.edu/10.1145/876638.876643>
- [90] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” in *2014 { USENIX } Annual Technical Conference ({ USENIX } { ATC } 14)*, 2014, pp. 305–319.
- [91] L. Lamport, “The TLA+ Video Course,” <https://lambort.azurewebsites.net/video/videos.html>.
- [92] M. A. Kuppe, L. Lamport, and D. Ricketts, “The tla+ toolbox,” *Electronic Proceedings in Theoretical Computer Science*, vol. 310, p. 50–62, Dec 2019. [Online]. Available: <http://dx.doi.org/10.4204/EPTCS.310.6>
- [93] “TLA+ on GitHub,” <https://github.com/tlaplus>.
- [94] M. Gario and A. Micheli, “Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms,” in *SMT workshop*, vol. 2015, 2015.