SAT-Based Quantified Symmetric Minimization of the Reachable States of Distributed Protocols

Katalin Fazekas  
TU Wien  
Vienna, Austria  
katalin.fazekas@tuwien.ac.at

Aman Goel  
Amazon Web Services  
Seattle, USA  
goelaman@amazon.com

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

Abstract—Most of the recent published work on the automated verification of distributed protocols has been concerned with deriving an inductive invariant that implies a safety specification. In this paper we argue that the inherent structural symmetry of protocols strongly suggests the existence of a unique property-independent formula \( r_{\text{min}} \) that describes a protocol’s reachable states as a minimum-cost conjunction of quantified first-order logic predicates. We show, for finite instances, that these predicates correspond to symmetry orbits of prime implicates, and show how they are derived using a novel SAT-based logic minimization algorithm which relies on the connection between symmetry and quantification as complementary ways of representing these orbits. We also present empirical data showing that the minimum-cost orbits derived for increasing protocol sizes converge syntactically, reaching a fixed point at a relatively small critical size. Our findings, thus, confirm earlier observations about the cutoff and saturation phenomenon of parameterized systems. To our knowledge, our approach is the first to algorithmically derive quantified first-order logic formulas for the reachable states of unbounded parameterized systems, enabling the verification of any safety property.

Index Terms—Distributed protocols, logic minimization, invariant inference, symmetry, quantifier inference.

I. INTRODUCTION

Driven by the availability of modern Satisfiability Modulo Theories (SMT) solvers [1], [2], the last few years have seen increasing interest in finding ways to automate the analysis and verification of distributed protocol specifications. Most of the recent published work [3]–[9] has been concerned with deriving an inductive invariant in quantified first-order logic (FOL) that serves as a proof certificate of a protocol’s safety property.

In this paper we argue that (an enhanced version of) classical logic minimization adds a new perspective that furthers our understanding of protocol behavior. Specifically we show, for a restricted class of protocol specifications, that it is possible to algorithmically derive a formula \( r_{\text{min}} \), that encodes the reachable states as an exact minimum-cost conjunction of quantified FOL invariants. For this purpose, we define the cost of a quantified invariant in prenex normal form (PNF) to be the sum of the number of quantifiers in its prefix and the number of literals in its matrix.

Key to deriving these minimum-cost formulas for the reachable states is the inherent structural symmetries of proto-

\(^1\)The fact that \( QSM \) and \( QM \) have two identical initials is purely coincidental.

\(^2\)Work does not relate to Aman Goel’s position at Amazon
The first empirical demonstration of the cutoff phenomenon for a collection of distributed protocols based on deriving a quantified FOL formula $r_{\text{min}}$ that encodes the protocol’s reachable states for all sizes.

The paper is organized as follows: Section II provides preliminaries and context. Section III details the QSM algorithm. Section IV shows how QSM, applied to increasing protocol sizes, reaches cutoff. Section V presents our experimental evaluation with Section VI giving a brief survey of related work. Section VII concludes the paper with future work directions.

II. Preliminaries and Context

We assume familiarity with the basics of 2-valued Boolean algebra including literals, minterms, prime implicants, and prime implicants of an $n$-variable function $f(x_1, \ldots, x_n)$ as well as basic notions from group theory including permutation groups, cycle notation, orbits, etc., which can be readily found in standard textbooks on Abstract Algebra [12]. We use $\text{primes}(f)$ to denote the disjunction of $f$’s prime implicants, i.e., its complete sum. On the other hand, the complete product\(^2\) of $f$ is the conjunction of its prime implicates and can be expressed as $\neg \text{primes}(\neg f)$. $\text{primes}(f)$ can also be viewed as a set and we define $\# \text{lits}(\rho)$ for $\rho \in \text{primes}(f)$ to be the number of literals in $\rho$.

A. Exact Two-Level Minimization

Exact two-level sum-of-product (SOP/DNF) minimization is an optimization problem seeking to find a minimum-cost subset of $\text{primes}(f)$ that covers all of $f$’s minterms. Mathematically, the problem can be stated as finding a Boolean assignment to a set of selector variables $z_\rho \in \{0, 1\}$, for $\rho \in \text{primes}(f)$, that represents a solution to the following set covering problem [13]:

$$\begin{align*}
\text{minimize} & \sum_{\rho \in \text{primes}(f)} \text{cost}(\rho) \times z_\rho \\
\text{subject to} & \left( \bigvee_{\rho \in \text{primes}(f)} z_\rho \land \rho \right) = f
\end{align*}$$

(1)

where $\text{cost}(\rho) \triangleq \# \text{lits}(\rho)$. This formulation can also be used to find a minimum-cost product-of-sums (POS/CNF) solution by applying De Morgan’s law to the minimum-cost SOP solution of $\neg f$.

B. The Quine-McCluskey Algorithm

The classical Quine-McCluskey (QM) algorithm [14]–[16] solves this problem by first deriving $\text{primes}(f)$ using a tabular procedure starting from $f$’s minterms, followed by a branch-and-bound search to find the optimal solution to (1). Both steps assume an explicit listing of $f$’s minterms. In particular, the set covering problem is represented as a 2-dimensional $\{0, 1\}$ prime implicant chart whose rows and columns correspond, respectively, to $f$’s minterms and prime implicants. A 1 (resp. 0) entry in row $\mu$ and column $\rho$ indicates that minterm $\mu$ is (resp. is not) covered by prime implicant $\rho$. In this encoding, the optimization objective is stated as finding a minimum-cost set of columns that covers all the rows.

C. Distributed Protocols

Our focus is the verification of distributed protocol specifications, i.e., protocols described at an abstraction level that hides code implementation details that model network topology and the effects of message interleaving, message loss, node failures, etc. Such specifications are typically encoded in FOL in such languages as TLA+ [17] or Ivy [18].

We specifically consider the class of multi-sorted data-independent protocol specifications [19], [20] that satisfy the following three requirements:

- The protocol sorts are unbounded sets of interchangeable structurally-symmetric elements.
- The protocol actions are atomic and asynchronous, i.e., they occur one at a time and interleave arbitrarily.
- The protocol encoding is in the empty theory of FOL, namely equality with uninterpreted functions.

This class encompasses a wide range of common protocols and should be considered a starting point that does not exclude future extensions to other types of protocols such as ones with totally-ordered sorts.

For purposes of illustration, and without loss of generality, in this paper we consider a protocol $P$ defined over a single unbounded sort $\text{node} \triangleq \{n_0, n_1, n_2, \ldots\}$ along with a) a finite set of relations\(^3\) on $\text{node}$ that serve as $P$’s state variables, and b) a finite set of actions that capture $P$’s state transitions. The elements of $\text{node}$ are referred to as its constants and are assumed to be indistinguishable; they can be arbitrarily permuted without changing $P$’s behavior. A predicate $\Psi$ on $P$’s state variables is a closed quantified FOL expression. In prenex normal form (PNF) it can be expressed as $\Psi \triangleq Q_1X_1Q_2X_2\cdots Q_nX_n. \psi(X_1, X_2, \ldots, X_n)$ where $Q_i \in \{\forall, \exists\}$, $X_i \in \text{node}$ and $\psi$ is a quantifier-free Boolean formula over $P$’s relations. Following standard practice, we define $\text{prefix}(\Psi)$ as the string of quantifiers and bound variables, and refer to $\psi$ as $\text{matrix}(\Psi)$. In the context of minimization, we further define the quantified cost of $\Psi$ as

$$q\text{Cost}(\Psi) = \#Q(\text{prefix}(\Psi)) + \#\text{lits}(\text{matrix}(\Psi))$$

(2)

where $\#Q$ is the number of $\Psi$’s quantified variables.

We use $P^k$ to denote a finite instance of $P$ defined over $\text{node}^k \triangleq \{n_0, n_1, \ldots, n_{k-1}\}$ for $k \geq 1$. Instantiating $P^k$’s relations with all possible combinations of its constants yields $P^k$’s state variables, denoted $\text{vars}^k$, whose cardinality is

$$|\text{vars}^k| = \sum_{h \in \text{relations}} k^{\text{arity}(h)}$$

(3)

$P^k$ is structurally symmetric; its behavior remains invariant under the action of $\text{Sym}(\text{node}^k)$, the group of permutations

\(^2\)Sum and product are commonly used in the hardware logic design literature. They are synonymous with disjunction and conjunction.

\(^3\)The arity of these relations is typically between 1 and 4.
TABLE I: Sample explicit clause orbits and their implicit encoding by finitely-quantified FOL formulas

<table>
<thead>
<tr>
<th>Partial Protocol Spec $\mathcal{P}$</th>
<th>Domain: node $\triangleq {n_0, n_1, n_2, \cdots}$</th>
<th>Relations: $a: node \rightarrow {0, 1}$, $b: node \rightarrow {0, 1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finite Protocol Instance $\mathcal{P}^3$</td>
<td>node$^3$ $\triangleq {n_0, n_1, n_2}$</td>
<td>vars$^3$ $= {a(n_0), a(n_1), a(n_2), b(n_0), b(n_1), b(n_2)}$</td>
</tr>
<tr>
<td>explicit clause orbit</td>
<td>$Sym(node^3)$: ${ (n_0n_1n_2), (n_0n_2n_1), (n_1n_0n_2), (n_1n_2n_0), (n_2n_0n_1), (n_2n_1n_0) }$</td>
<td></td>
</tr>
<tr>
<td>implicit-quantified orbit</td>
<td>$\forall^3 N, M : (N = M) \lor a(N) \lor b(M)$</td>
<td>$qCost = 2 + 3 = 5$</td>
</tr>
<tr>
<td>a) $\forall^3$ quantification</td>
<td>$\exists^3 N : b(N)$</td>
<td>$qCost = 1 + 1 = 2$</td>
</tr>
<tr>
<td>b) $\exists^3$ quantification</td>
<td>$\forall^3 N, \exists^3 M : a(N) \lor b(M)$</td>
<td>$qCost = 2 + 2 = 4$</td>
</tr>
<tr>
<td>c) Mixed $\forall^3\exists^3$ quantification</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

on a $k$-element set. In particular, $Sym(node^k)$ partitions $\mathcal{P}^k$’s variables, as well as any Boolean expressions on them (conjunctions, disjunctions, etc.) into equivalence classes or orbits. Given any finite set $S^k$ of syntactically “similar” formulas on the variables of $\mathcal{P}^k$ and any $f \in S^k$ we define

$$orbit^k(f) \triangleq \{ g \in S^k | \exists \pi \in Sym(node^k) : \pi(f) = g \} \quad (4)$$

where $\pi(f)$ is the result of applying the permutation $\pi$ to $f$. The set of orbits in $S^k$ will be denoted as $orbs(S^k)$.

We are particularly interested in clausal orbits and their compact encoding as finitely-quantified FOL predicates. Table I illustrates this concept with three example clausal orbits. We assume that our generic protocol $\mathcal{P}$ has two unary relations labeled $a$ and $b$. Its finite instantiation with 3 nodes creates 6 variables and has $3! = 6$ structural symmetries (identity, 3 swaps, and 2 rotations) expressed as node permutations in standard cycle notation. The example orbits in the table represent a 6-clause orbit in column (a), a 1-clause orbit in column (b), and a 3-clause orbit in column (c). Note that the set of clauses in each of these orbits remains unchanged under the action of the 6 permutations of $node^3$. The effect of these permutations is to simply reorder the literals and clauses in each orbit while preserving logical equivalence. Logical invariance, in fact, is a direct consequence of two properties of conjunction and disjunction: idempotency ($x \land x = x$, $x \lor x = x$) and commutativity ($x \land y = y \land x$, $x \lor y = y \lor x$).

The last row in Table I shows the finitely-quantified FOL formulas that encode these clause orbits. To emphasize that the quantification is over the finite node$^3$ set, we use the convention of annotating the universal and existential quantifiers with a “3” superscript. Each of these formulas are derived by the mechanical quantifier inference procedure from [6]. This procedure is based on a syntactic analysis of any clause in the orbit (basically the number and distribution of sort constants in the clause’s relations) and guarantees that instantiating the universal and existential quantifiers in these formulas over node$^3$ yields the exact set of clauses in the corresponding explicit orbits, except possibly for potential duplicates and tautologies. The correspondence between an explicit orbit $orbit^k$ and its finitely-quantified encoding $\Psi^k_i$ can be expressed by a pair of related functions as

$$\Psi^k_i = qInf(orbit^k_i)$$

$$\text{orbit}^k_i = qIns(\Psi^k_i) \quad (5)$$

where $qInf$ performs finite quantifier inference whereas $qIns$ performs finite quantifier instantiation.

D. An Example $r_{\min}$ Formula

Before describing the steps for deriving $r_{\min}$, let’s illustrate it for a specific example. Consider the TLA+ specification [21] of the Transaction Commit (TC) protocol [22]. This protocol is based on a single sort for representing resource managers and four unary relations working, prepared, committed, and aborted. Denoting these relations by their initials, the minimum formula produced by QSM for the protocol’s reachable states converged syntactically at a finite instance with 2 resource managers yielding the following eight-orbit expression:

$$r_{\min}(TC) = \bigwedge_{1 \leq i \leq 8} \Psi_i$$

$$\Psi_1 = \forall R.(a(R) \rightarrow \neg w(R))$$

$$\Psi_2 = \forall R.(a(R) \rightarrow \neg p(R))$$

$$\Psi_3 = \forall R.(a(R) \rightarrow \neg c(R))$$

$$\Psi_4 = \forall R.(p(R) \rightarrow \neg w(R))$$

$$\Psi_5 = \forall R.(c(R) \rightarrow \neg p(R))$$

$$\Psi_6 = \forall R.(c(R) \rightarrow \neg w(R))$$

$$\Psi_7 = \forall R.(w(R) \lor p(R) \lor c(R) \lor a(R))$$

$$\Psi_8 = \forall R_1, R_2.$$  

$c(R_1) \land \neg c(R_2) \land \neg p(R_2) \rightarrow (R_1 = R_2)$

An unbounded quantified SMT query showed that this formula is indeed an inductive invariant for TC. Checking if TC satisfies any desired safety property $S$ can now be achieved by showing that $r_{\min}(TC) \rightarrow S$ is valid. More interestingly, the shortest strengthening assertion that explains why $S$ holds can be seen as a minimal subset of the eight orbits in (6). Denoting this subset by $A_{\min}(TC, S)$ it can be found using a minimal unsatisfiable subset (MUS) extractor, such as MARCO [23], from the UNSAT CNF formula

$$A_{\min}(TC, S) = MUS[(r_{\min}(TC) \land S) \land T \land (r'_ {\min}(TC) \land S')]$$
where $T$ is the transition relation, the primes indicate a variable’s next state, and the clauses of $r_{min}(TC)$ are highlighted to emphasize that they are treated as soft clauses by the MUS extractor. For example, given the following two safety properties for TC.

$$S_1 = \forall R_1, R_2. (\neg a(R_1) \lor \neg c(R_2))$$
$$S_2 = \forall R_1, R_2. (\neg w(R_1) \lor \neg c(R_2))$$

we can show that their shortest respective proof certificates/strengthening assertions are:

$$A_{min}(TC, S_1) = \Psi_2$$
$$A_{min}(TC, S_2) = \Psi_4$$

III. QSM: SAT-BASED QUANTIFIED SYMMETRIC MINIMIZATION

The QSM minimization algorithm seeks to derive a minimum-cost finitely-quantified formula for $r^k$, the set of reachable states of $P^k$. To achieve this, it takes advantage of two features of these formulas. The first, obvious, feature is the structural symmetry of $P^k$. QSM preserves this symmetry by operating on prime implicant orbits rather than on individual prime implicates. The second, less obvious, feature is that the number of $P^k$’s reachable states is almost always much smaller than the number of its unreachable states. This suggests seeking a minimum-cost CNF, rather than DNF, solution. Before delving into the detailed description of QSM, it is helpful to understand its operation at a very high level as

$$P^k \xrightarrow{QSM} r^k_{min} = \bigwedge_{1 \leq i \leq l} \psi^k_i \quad (7)$$

In other words, QSM produces a minimum-cost conjunction of $l$ finitely-quantified FOL formulas where each $\psi^k_i$ captures an orbit of $r^k$’s prime implicants.

In contrast to (1), the minimization problem for $r^k$ can now be stated as finding a Boolean assignment to a set of selector variables $z_{i, \omega}$, for $\omega^k \in orbs(primes(\neg r^k))$, that represents a solution of the set covering problem

$$\min_{\omega^k \in orbs(primes(\neg r^k))} \sum_{\omega^k} qCost(qInf(\neg \omega^k)) \times z_{i, \omega^k}$$

s.t.

$$\bigvee_{\omega^k \in orbs(primes(\neg r^k))} (z_{i, \omega^k} \land \omega^k) = \neg r^k \quad (8)$$

Viewed as a formula, each such orbit $\omega^k$ is a disjunction of symmetric prime implicates; thus, its negation $\neg \omega^k$ is a conjunction of symmetric prime implicates, i.e., a clausal orbit. This explains the particular choice of the cost metric in (8).

The derivation of $r^k_{min}$ in QSM is a deterministic mechanical procedure consisting of the following four steps:

1) A BDD-based forward image computation [24] to produce a DNF representation of $r^k$.
2) A SAT-based procedure to generate the set of prime implicant orbits of $\neg r^k$.
3) A quantifier-inference procedure $qInf$ from [6] that outputs a finitely-quantified FOL formula for each prime implicate orbit of $r^k$ along with its $qCost$.
4) A branch-and-bound set covering procedure that finds the minimal number of prime implicate orbits that cover $r^k$ using their quantified cost as the minimization objective.

A. Symmetry-Aware Enumeration of Prime Implicant Orbits

The PI enumeration algorithm operates on the CNF formula representing $\neg r^k$ and is based on a dualRail encoding of the state variables [25, 26]. Specifically, each state variable $x$ is encoded using two fresh variables $x^p$ and $x^n$ according to

$$\begin{array}{c|c|c}
 x^p & x^n & \text{X} \\
 \hline
 0 & 0 & d \\
 0 & 1 & 0 \\
 1 & 0 & 1 \\
 1 & 1 & \text{invalid}
\end{array}$$

where $d$ stands for don’t-care. The dualRail version of $\neg r^k$ is obtained by replacing all positive (resp. negative) appearances of $x$ with $x^p$ (resp. $x^n$) and by adding the clause $(\neg x^p \lor \neg x^n)$ to exclude the invalid combination. This encoding is reversible: given any conjunction (model) or disjunction (clause) of $\neg r^k$ we use dualRail($\neg r^k$) to denote the above encoding, and singleRail(dualRail($\neg r^k$)) to recover the $\neg r^k$ formula based on the original state variables.

This encoding makes it possible to interpret the complete assignments produced by a SAT solver for the $x^p$ and $x^n$ variables as partial assignments (i.e., assignments with don’t cares) for the original $x$ variables. Assuming that $|vars^k| = m$, a prime implicant consisting of $l$ literals corresponds to (i.e., covers) $2m-l$ states and can be found by checking the satisfiability of the conjunction of dualRail($\neg r^k$) with the following pseudo-Boolean (cardinality) constraint [27]:

$$\sum_{1 \leq i \leq m} (x^p_i + x^n_i) \leq l \quad (9)$$

The orbit enumeration procedure is depicted in Algorithm 1. The procedure accepts a CNF representation of $\neg r^k$ and returns the complete set of prime orbits. The primes are found, in increasing literal size, by executing the SAT query
on line 5 for \(i = 1, \cdots, m\) using a single incremental SAT solver instance based on an incremental encoding [28] of the cardinality constraint (9). If satisfiable, the solution to the query is an \(i\)-literal prime \(\rho_D\) in dualRail encoding. The orbit of the singleRail encoding of this prime, computed by applying the appropriate structural symmetry permutations to its sort constants (line 7), is then added to primeOrbits (line 8) and eliminated from further consideration (line 9) for all subsequent SAT queries. When the query is unsatisfiable (i.e., when there are no \(i\)-literal primes or all \(i\)-literal primes have been found), \(i\) is incremented to find primes with \(i + 1\) literals.

B. Symmetry-Aware Set Covering

Our QSM algorithm is an adaptation of the standard textbook branch-and-bound (BnB) logic minimization procedure that uses an explicit matrix encoding of the covering constraints. Specifically, it is based on the BCP procedure for unate and binate covering in [29]. This procedure has three parts: a) a reduction step that uses column and row dominance rules to identify essential and covered (dominated) primes, b) a termination check to accept or reject a complete solution by comparing its cost to the best seen so far, and c) a depth-first BnB search when the “reduced” covering constraints become cyclic. The QSM algorithm closely follows this computational flow but replaces the column and row dominance rules with queries to an incremental SAT solver using an implicit CNF encoding of the covering constraints.

To simplify the description of QSM, let’s assume that the prime orbits are numbered from 1 to \(n\), i.e., \(\text{orbs}(\text{primes}(-r^k)) = \{\omega^k_1, \cdots, \omega^k_n\}\), and let \([n] \triangleq \{1, 2, \cdots, n\}\). The covering constraints can now be captured by the CNF formula

\[
\varphi^k \triangleq \bigwedge_{i \in [n]} (\neg z_i \lor \neg \omega^k_i)
\]

which can be queried by an incremental SAT solver under different assumptions involving the literals of the formula. Specifically, the SAT query

\[
\text{SAT}\{[\varphi^k, \text{assume } \text{chosenLiterals}(\varphi^k)]\}
\]

checks the satisfiability of \(\varphi^k\) assuming that all literals in \(\text{chosenLiterals}(\varphi^k)\) are set to True. These literals can include the protocol state variables as well as the selection variables. In particular, it is convenient to define the orbit selection formula

\[
Z(\text{sel}) \triangleq (\bigwedge_{i \in \text{sel}} z_i) \land (\bigwedge_{i \notin \text{sel}} \neg z_i)
\]

which can serve as an assumption in (11) to activate the prime orbits specified by the set \(\text{sel} \subseteq [n]\) and to deactivate the remaining orbits.

During the search we use \(\text{sol} \subseteq [n]\) to represent the set of prime orbits in the current partial solution and \(\text{pnd} \subseteq [n]\) for the prime orbits that are pending, i.e., the orbits that may or may not be needed to complete the solution. \(\text{sol}\) becomes a complete solution when \(\text{pnd} = \emptyset\).

Identifying Essential Orbits: A pending prime orbit \(\omega^k_i\) is essential if it covers some states that are not covered by the union of a) the remaining pending orbits and b) the orbits in the current partial solution; otherwise it is not essential. This can be checked by the SAT query

\[
\text{isEssential}(\omega^k_i) \triangleq \text{SAT}\{\neg(\omega^k_i \to \bigvee_{j \in \text{sol} \cup \text{pnd}\setminus \{i\}} \omega^k_j)\}
\]

\[
= \text{SAT}\{\omega^k_i \land \bigwedge_{j \in \text{sol} \cup \text{pnd}\setminus \{i\}} \neg \omega^k_j\}
\]

Since \(\omega^k_i\) is a disjunction of primes, the formula in this query is not in CNF. By symmetry, however, it is sufficient to check the essentiality of any prime \(\rho \in \omega^k_i\) to conclude if the whole orbit is or is not essential. This allows the above query to be re-expressed as

\[
\text{isEssential}(\omega^k_i) = \text{SAT}\{\rho(\omega^k_i) \land \bigwedge_{j \in \text{sol} \cup \text{pnd}\setminus \{i\}} \neg \omega^k_j\}
\]

where, with a slight notational abuse, \(\rho(\omega^k_i)\) is used to assert an arbitrary prime (a conjunction of protocol literals) from the \(\omega^k_i\) orbit. The SAT query to check whether or not the \(\omega^k_i\) orbit is essential can now be expressed as

\[
\text{isEssential}(\omega^k_i) = \text{SAT}\{[\varphi^k, \text{assume } \rho(\omega^k_i) \land Z(\text{sol} \cup \text{pnd}\setminus \{i\})]\}
\]

Identifying Covered and Partially-Covered Orbits: The coverage of a pending orbit is the number of states it covers that are not already covered by the current partial solution and can be found as the solution of this \#SAT [30] query:

\[
\text{coverage}(\omega^k_i) \triangleq \#\text{SAT}\{\neg(\omega^k_i \to \bigvee_{j \in \text{sol}} \omega^k_j)\}
\]

Exact coverage can, thus, be expressed as

\[
\text{coverage}(\omega^k_i) = \#\text{SAT}\{[\varphi^k, \text{assume } \rho(\omega^k_i) \land Z(\text{sol})]\}
\]

Coverage is used to remove completely covered orbits from \(\text{pnd}\) (when \(\text{coverage} = 0\)) and to rank partially-covered orbits for the branching step (when \(\text{coverage} > 0\)). Our implementation uses an approximation of \#SAT since the exact number of solutions to (14) is not needed. The coverage estimate of pending orbits is stored in an array \(\text{cov}\).

The pseudo-code of QSM is shown in Algorithm 2. Initially, \(\text{pnd} = [n], \text{sol} = \emptyset\), the entries in the \(\text{cov}\) array are uninitialized, and \(\text{UB}\) (the upper bound on the cost of the solution) is set to \(1 + \sum_{i \in [n]} q\text{Cost}(\omega^k_i)\).

At each invocation, QSM performs the following steps:

- Line 2: It updates the current covering requirements (encoded by \(\text{pnd}, \text{sol}\), and \(\text{cov}\)) by calling reduce to identify essential and covered primes, if any.
- Lines 3-8: It checks if a complete solution has been found and
  - Lines 4-6: returns this solution and updates \(\text{UB}\) to its cost if it is cheaper than the best seen so far.
Algorithm 2 Quantified Symmetric Minimization

1. procedure QSM(pnd, sol, cov, UB)
2. (pnd, sol, cov) ← reduce(pnd, sol, cov)
3. if pnd = ∅ then
4. if qCost(sol) < UB then
5. UB ← qCost(sol)
6. return sol
7. return NoSolution
8. if LB ≥ UB then
9. return NoSolution
10. i ← chooseOrbit(pnd, cov)
11. S^{with}_{i} ← QSM(pnd \ {i}, sol ∪ {i}, cov, UB)
12. if qCost(S^{with}_{i}) = LB then
13. return (S^{with}_{i})
14. S^{without}_{i} ← QSM(pnd \ {i}, sol, cov, UB)
15. return BestSolution(S^{with}_{i}, S^{without}_{i})
16. procedure reduce(pnd, sol, cov)
17. (existEss, pnd, sol) ← addEssentials(pnd, sol)
18. (existCov, pnd, cov) ← removeCovered(pnd, sol, cov)
19. if existEss ∨ existCov then
20. (pnd, sol, cov) ← reduce(pnd, sol, cov)
21. return (pnd, sol, cov)
22. procedure addEssentials(pnd, sol)
23. essentials ← ∅
24. for each orbit ∈ pnd do
25. if isEssential(orbit, pnd, sol) then
26. essentials ← essentials ∪ {orbit}
27. sol ← sol ∪ essentials
28. pnd ← pnd \ essentials
29. return (essentials > 0, pnd, sol)
30. procedure removeCovered(pnd, sol, cov)
31. covered ← ∅
32. for each orbit ∈ pnd do
33. cov[orbit] ← coverage(orbit, sol)
34. if cov[orbit] = 0 then
35. covered ← covered ∪ {orbit}
36. pnd ← pnd \ covered
37. return (covered > 0, pnd, cov)

− Lines 7-8: returns “no solution” (i.e., backtracks) if the cost is higher than the best seen so far.
− Lines 9-11: It sets the lower bound LB to be the cost of the current partial solution and backtracks if that cost is greater than the current upper bound.
− Line 12: It ranks the pending orbits by their estimated coverage and chooses the orbit with the highest coverage for the next branching decision breaking ties arbitrarily. In addition, it ranks orbits that are not parameterized by sort constants (i.e., they are independent of “k”) higher than other orbits. The intuition behind this heuristic is that such orbits are more likely to be in the minimum solution since they are size-independent.
− Lines 13-17: It recursively calls itself to search for a solution that includes the chosen orbit and returns that solution if its cost is equal to the lower bound. Otherwise, it recursively calls itself to search for a solution that excludes the chosen orbit and returns the cheaper of the two solutions.

The computational core of QSM is in the reduce, addEssentials, and removeCovered procedures. The reduce procedure repeatedly calls addEssentials and removeCovered until all essential and covered orbits have been processed and pnd, sol, and cov updated. Finally, the addEssentials and removeCovered procedures implement the SAT queries corresponding to (13) and (14).

IV. FROM BOUNDED TO UNBOUNDED MINIMIZATION

Applying the QSM algorithm to $\mathcal{P}^{1}, \mathcal{P}^{2}, \cdots$ generates a corresponding sequence of minimum solutions $r_{\min}^{k}, r_{\min}^{k+1}, \cdots$. An interesting empirical observation is that this sequence reaches a syntactic fixed point (Figure 1) at some value $k^{*}$ defined as:

1. $r_{\min}^{k^{*}} = \bigwedge_{1 \leq i \leq l} \Psi_{i}^{k^{*}}$
2. $r_{\min}^{k^{*}+1} = \bigwedge_{1 \leq i \leq l} \Psi_{i}^{k^{*}+1}$
3. $\forall i \in [1, l]: prefix(\Psi_{i}^{k^{*}+1}) = prefix(\Psi_{i}^{k^{*}})$
4. $\forall i \in [1, l]: matrix(\Psi_{i}^{k^{*}+1}) = matrix(\Psi_{i}^{k^{*}})$

Another way of saying this is that the minimum clausal orbits “converge” and that additional orbits that might be produced at values of $k$ larger than $k^{*}$ become redundant and do not introduce new behaviors beyond $k^{*}$. This is reminiscent of the cutoff [31], [32] and data saturation [20] phenomena in the model checking literature and suggests that the finite quantification can be replaced with unbounded quantification yielding an exact minimum formula

$$r_{\min} = \bigwedge_{1 \leq i \leq l} \Psi_{i} \quad (16)$$

for the unbounded protocol $\mathcal{P}$. Our contribution can be seen as the culmination of these earlier efforts by showing that the incorporation of minimization a) yields the natural quantified forms of the $r_{\min}$ orbits and b) “explains” how saturation happens.

V. EXPERIMENTAL EVALUATION

We evaluated QSM on a set of 17 protocols from [4], [5], [33]. This set includes fairly complex high-level descriptions of

4In practice, the initial base size usually starts at $i > 1$. 

![Fig. 1: QSM Syntactic Fixed Point Convergence](image-url)
TABLE II: QSM Experimental Results†

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Memory MB</th>
<th>CPU Time, sec</th>
<th>Number of</th>
<th>$r_{min}$</th>
<th>Assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Total</td>
<td>BDD</td>
<td>PI Gen</td>
<td>qInf</td>
</tr>
<tr>
<td>tla-consensus</td>
<td>59</td>
<td>9</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>tla-tecommit</td>
<td>67</td>
<td>9</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>tla-twophase</td>
<td>67</td>
<td>7197</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>distai-ricart-agrawnala</td>
<td>67</td>
<td>9</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>i4-lock-server</td>
<td>67</td>
<td>17</td>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>pyv-shared-kv</td>
<td>104</td>
<td>58</td>
<td>22</td>
<td>10</td>
<td>17</td>
</tr>
<tr>
<td>pyv-shared-kv-no-loss-keys</td>
<td>106</td>
<td>68</td>
<td>22</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>ex-simple-decentralized-lock</td>
<td>67</td>
<td>18</td>
<td>8</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>pyv-firewall</td>
<td>67</td>
<td>56</td>
<td>6</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>pyv-lockserv</td>
<td>67</td>
<td>9</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>pyv-consensus-automaton</td>
<td>67</td>
<td>9</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>ex-toy-consensus</td>
<td>108</td>
<td>29</td>
<td>14</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ex-naive-consensus</td>
<td>67</td>
<td>21</td>
<td>9</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>ex-simple-election</td>
<td>47056</td>
<td>3812</td>
<td>3773</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>pyv-toy-consensus-forall</td>
<td>67</td>
<td>31</td>
<td>13</td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>pyv-toy-consensus-epr</td>
<td>48912</td>
<td>3622</td>
<td>3607</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>pyv-consensus-epr</td>
<td>913</td>
<td>7401</td>
<td>4675</td>
<td>74</td>
<td>32</td>
</tr>
</tbody>
</table>

† The memory and time statistics capture the runs of QSM from the initial finite size to the one larger than the cutoff size. The number of variables, cubes, Pls, and Orbits are at the cutoff size.

*For these protocols, $r_{min}$ was found without any branch-and-bound search.

We can make the following observations about these results.

- Except for 4 cases, the total time for deriving $r_{min}$ is less than a couple of CPU minutes.
- For 8 protocols, $r_{min}$ was found without any branch-and-bound search (indicated with * in the Cov column).
- Except for the tla-twophase protocol, the derivation of $r_{min}$ was completed and the solution was unique. The minimization step timed out for tla-twophase. Interestingly though, the complete set of orbits at sizes 2 and 3 were identical and found to be inductive. Thus, even in this case the complete product was unique.
- In 3 cases, the BDD image computation had a large memory footprint and dominated the total run time.

We can make the following observations about these results.

- Except for 4 cases, the total time for deriving $r_{min}$ is less than a couple of CPU minutes.
- For 8 protocols, $r_{min}$ was found without any branch-and-bound search (indicated with * in the Cov column).
- Except for the tla-twophase protocol, the derivation of $r_{min}$ was completed and the solution was unique. The minimization step timed out for tla-twophase. Interestingly though, the complete set of orbits at sizes 2 and 3 were identical and found to be inductive. Thus, even in this case the complete product was unique.
- In 3 cases, the BDD image computation had a large memory footprint and dominated the total run time.

A preliminary analysis of these results identified the causes for the observed computational bottlenecks. Specifically, the current BDD front-end does not account for symmetry causing a huge memory blow-up and the attendant increase in run time. A natural solution would be to preserve the protocol’s structural symmetry in the forward image computation in order to produce a set of cube orbits, rather than individual cubes. Specifically, the excessive run time of the covering step in tla-twophase and pyv-consensus-epr was a direct consequence of the large number of Pls and PI orbits and the failure of the branching heuristic to identify good candidate orbits that can guide the search to close-to-minimal initial solutions. We noticed that many of the PI orbits in these, as well as other, protocols are “similar” (involving the same literals) and can be merged as disjoint sub-orbits into larger super orbits. As an example, Table IV shows 5 sub-orbits from the ex-simple-decentralized-lock protocol and their merger into a single super orbit with a much smaller $qCost$. Identifying super orbits

mutual-exclusion and consensus algorithms, including protocols such as sharded key-value store, two-phase commit, asynchronous lock server, Ricart-Agrawala, etc. Several studies [4], [5], [18], [34]–[36] have indicated the challenges involved in verifying these protocols.

We assessed the performance of each step in QSM and contrasted its derivation of $r_{min}$ (which is inferred independently of any protocol property) to human-written and automatically-derived property-driven strengthening assertions. For each protocol in Table II we made a sequence of QSM runs from an initial base size to the converged $k^*$ cutoff size (details on these sizes are shown in Table III) and report the cumulative time and maximum memory usage for all these runs. The total time for these runs is broken down into the following stages:

- **BDD**: the time to generate the DNF table (as a set of cubes) of $r^k$ using BDD-based forward image computation.
- **PI Gen**: the time for the SAT-based procedure to enumerate the prime implicants of $\neg r^k$ and to partition them into symmetry orbits.
- **qInf**: the time to perform quantifier inference on all prime implicate orbits.
- **Cov**: the time for the SAT-based branch-and-bound set covering minimization problem that yields $r_{min}^k$.

The table also shows, at the cutoff size, the number of variables, cubes, prime implicants/implicates, and orbits. Column “$r_{min}$ Orbits” gives the number of (invariant) orbits in the final unbounded formula $r_{min}$; these formulas were independently confirmed to be inductive using Ivy [18]. Note that $r_{min}$ can be instantiated for any arbitrary protocol size $k$ and can be independently confirmed to be logically-equivalent to the set of reachable states of $P^k$. The final two columns give the number of manually-written and automatically-derived strengthening assertions using IC3PO [6], [37] for the protocol’s specified safety property.

The current BDD front-end limited the set of protocols our prototype can support.
finite instance sizes

TABLE III: Finite instance sizes from the initial base size to the converged cutoff size

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Finite instance sizes † ‡</th>
</tr>
</thead>
<tbody>
<tr>
<td>tla-consensus</td>
<td>value = 2 ↦ 3</td>
</tr>
<tr>
<td>tla-tcommit</td>
<td>resource-manager = 2</td>
</tr>
<tr>
<td>tla-twophase</td>
<td>resource-manager = 2</td>
</tr>
<tr>
<td>distai-ricart-agrawala</td>
<td>node = 2</td>
</tr>
<tr>
<td>i4-lock-server</td>
<td>client = 2 ↦ 3, server = 1 ↦ 2</td>
</tr>
<tr>
<td>pyv-sharded-kv</td>
<td>key = 2, node = 2 ↦ 3, value = 2 ↦ 3</td>
</tr>
<tr>
<td>pyv-sharded-kv-no-lost-keys</td>
<td>key = 2, node = 2 ↦ 3, value = 2 ↦ 3</td>
</tr>
<tr>
<td>ex-simple-decentralized-lock</td>
<td>node = 2 ↦ 3</td>
</tr>
<tr>
<td>pyv-firewall</td>
<td>node = 2 ↦ 3</td>
</tr>
<tr>
<td>pyv-lockserv</td>
<td>node = 2 ↦ 3</td>
</tr>
<tr>
<td>ex-lockserv-automaton</td>
<td>node = 2 ↦ 3</td>
</tr>
<tr>
<td>ex-toy-consensus</td>
<td>node = 2 ↦ 4, quorum = 1 ↦ 4, value = 2 ↦ 3</td>
</tr>
<tr>
<td>ex-naive-consensus</td>
<td>node = 3 ↦ 4, quorum = 3 ↦ 4, value = 3</td>
</tr>
<tr>
<td>ex-simple-election</td>
<td>acceptor = 2 ↦ 3, proposer = 2 ↦ 3, quorum = 1 ↦ 3</td>
</tr>
<tr>
<td>pyv-toy-consensus-forall</td>
<td>node = 2 ↦ 4, quorum = 1 ↦ 4, value = 2 ↦ 3</td>
</tr>
<tr>
<td>pyv-toy-consensus-epr</td>
<td>node = 2 ↦ 3, quorum = 1 ↦ 3, value = 2 ↦ 3</td>
</tr>
<tr>
<td>pyv-consensus-epr</td>
<td>node = 2 ↦ 4, quorum = 1 ↦ 4, value = 2 ↦ 3</td>
</tr>
</tbody>
</table>

† a = x denotes sort a has both initial size and final cutoff size x
‡ a = x ↦ y denotes sort a has initial size x and final cutoff size y

during the PI generation step will yield a much smaller number of orbits for the subsequent cover minimization step. These initial results provide strong support to our thesis that the structural symmetries of a protocol enable the derivation of a minimal conjunctive FOL formula for its reachable states.

VI. RELATED WORK

Notwithstanding the undecidability result of Apt and Kozen [38], many efforts to automatically infer quantified inductive invariants for distributed protocols have been reported with the pace increasing in recent years [3]–[9]. All these works, however, perform a property-dependent analysis of the distributed protocol and aim to derive an inductive invariant specific to a given safety property. In contrast, our work attempts to derive an FOL encoding of the exact set of reachable states of a distributed protocol, which can be utilized to check the validity of any safety property.

Several manual or semi-automatic verification techniques based on interactive theorem proving have been proposed for deriving system-level proofs [18], [34], [39]–[43]. However, unlike fully-automatic verification, all these methods require a detailed understanding of the intricate inner workings of the protocol and entail significant manual effort to guide proof development.

Verification of parameterized systems using SMT solvers is further explored in MCMT [44], Cubicle [45], and paraVer-ifer [46]. Our work is closest in spirit to view abstraction, proposed in [47], which computes the reachable set for finite instances using forward reachability until cutoff is reached. Our technique further builds on these works with the ability to automatically derive a quantified FOL encoding of the set of reachable states by utilizing a novel symmetry-aware SAT-based logic minimization algorithm.

In the context of logic minimization, the implicit encoding of the covering constraints in QSM is similar, at least in spirit but not details, to the procedure in [48]. Finally, it is worth noting, as an interesting historical fact, that McCluskey [16] considered the incorporation of Boolean symmetry in his tabular method for deriving the set of prime implicants.

VII. CONCLUSIONS AND FUTURE WORK

We proposed QSM, a novel forward-reachability algorithm that combines the relationship between symmetry and quantification in a SAT-based logic minimization procedure to derive a compact quantified FOL formula $r_{min}$ representing the set of reachable states of a distributed protocol. We empirically demonstrate the ability of our prototype to derive such quantified representations of the reachable states, independent of the protocol size, on a restricted class of distributed protocols. The derivation of $r_{min}$ is property-independent, enables checking the validity of any protocol safety property and compactly summarizes all protocol behaviors for any size.

In its current form, our QSM prototype is limited to protocol specifications based on unbounded symmetric sorts. Structural symmetry is a manifestation of what can be called spatial regularity which leads to boundedness in the spatial dimension.

TABLE IV: Illustrating the merger of sub-orbits into a single super orbit for the ex-simple-decentralized-lock protocol

<table>
<thead>
<tr>
<th>Sub-orbits</th>
<th>Equivalent Super Orbit</th>
</tr>
</thead>
<tbody>
<tr>
<td>invariant [pi19] for all N1, N2, N3.</td>
<td>(~has_lock(N1)</td>
</tr>
<tr>
<td>invariant [pi25] for all N1, N2.</td>
<td>(~has_lock(N1)</td>
</tr>
<tr>
<td>invariant [pi31] for all N1, N2.</td>
<td>(~has_lock(N1)</td>
</tr>
<tr>
<td>invariant [pi37] for all N1, N2.</td>
<td>(~has_lock(N1)</td>
</tr>
<tr>
<td>invariant [pi43] for all N.</td>
<td>(~has_lock(N)</td>
</tr>
</tbody>
</table>
An important extension would be to derive $r_{\text{min}}$ for protocols that also include totally-ordered sorts. We do not foresee conceptual difficulties for such an extension since totally-ordered sorts introduce another type of regularity, namely temporal regularity which leads to boundedness in the temporal dimension, as explored in [49] and applied to automatically prove the safety of Paxos [50] and Bakery [51] protocols. Intuitively, while a totally-ordered sort causes the state space of the protocol to expand without bound, that expansion must be characterized by a repeating pattern since, otherwise, it would not be captured by a finite set of quantifiers. Thus, as observed in [49], we expect a cutoff/saturation phenomenon conceptually similar to that exhibited by symmetry but different in implementation details. We also plan to augment QSM with the MARCO MUS extractor [23] to automatically derive subsets of the minimum orbits of $r_{\text{min}}$ that can serve as minimum strengthening assertions for given safety properties.

The experimental results strongly hint that the $r_{\text{min}}$ formula produced by QSM is unique. We conjecture that this must follow from symmetry and the particular cost function used in the set covering step. However, we do not have a formal proof that this is always the case and we plan to develop such a proof since solution uniqueness is critical for syntactic convergence. More speculatively, the possibility that a unique quantified formula for $r_{\text{min}}$ can be mechanically derived, even when it contains predicates that violate known decidable FOL classes, suggests perhaps the existence of a new decidable fragment of FOL.

Finally, the limited experiments we reported highlighted the need for several optimizations to our current prototype implementation of QSM including the incorporation of symmetry in BDD-based forward image computation, the identification of super orbits in the PI enumeration step, and improving the accuracy of coverage estimates during the branch-and-bound search for the minimum cover. Specifically, one simple modification of the #SAT query in (14) is to multiply its answer by the number of primes in the orbit to get a more accurate orbit coverage.

ACKNOWLEDGMENTS

This work was done in part while the authors were participating in a program at the Simons Institute for the Theory of Computing. The research was funded in part by the Austrian Science Fund (FWF) under project No. T-1306.

REFERENCES


