Formal Methods in Computer-Aided Design 2023

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_{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_{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 protocol specifications as well as the recently-established connection between symmetry and quantification [6]. Applied to finite protocol instances, our proposed ***Q**uantified **S**ymmetric **M**inimization (QSM)* algorithm preserves these symmetries in both the prime implicant (PI) generation and set covering phases of the classical Quine-McCluskey (QM)[1] algorithm. In addition, it replaces the unscalable tabular procedures in QM with scalable alternatives based on incremental SAT solving [10], [11]. Empirically, we also show that the finitely-quantified reachable state formulas generated by *QSM* at increasing protocol sizes reach a *syntactic fixed point* at a critical *cutoff* size and yield the minimum formula for the reachable states of the unbounded protocol. We believe this to be a direct consequence of the restrictions (elaborated later) on the class of protocols we consider, but leave a rigorous formal proof as future work.

The invariants in the r_{min} formula will be shown to be *prime implicate symmetry orbits* of the reachable states and that they represent the complete set of strengthening assertions needed to establish the validity of *any* safety property S. Intuitively, if $r_{min} \rightarrow S$ is valid, then S holds. However, the simplest explanation of why S holds might be a minimal subset of r_{min}'s orbits that acts as its strongest strengthening assertion.

Our key contributions include:

- A novel symmetry-aware SAT-based logic minimization algorithm that utilizes structural symmetry and its connection with quantification to derive an exact minimum-cost representation of the original structurally-symmetric formula as a finitely-quantified FOL formula.
- A novel forward-reachability algorithm that derives the strongest complete property-independent quantified formula r^k_{min} representing the set of reachable states for a protocol instance of size k.
- A simple property-independent procedure that derives $r^1_{min}, r^2_{min}, \ldots$ for protocol instances with increasing sizes until reaching convergence at a critical cutoff size k^*, where $r^{k^*}_{min}$ syntactically converges to $r^{k^*+1}_{min}$. At the cutoff size, r_{min} represents the strongest and complete inductive invariant that summarizes all protocol behaviors for any size.

[§]Work does not relate to Aman Goel's position at Amazon

[1]The fact that *QSM* and QM have two identical initials is purely coincidental.

https://doi.org/10.34727/2023/isbn.978-3-85448-060-0_23

This article is licensed under a Creative Commons Attribution 4.0 International License

- The first empirical demonstration of the cutoff phenomenon for a collection of distributed protocols based on deriving a quantified FOL formula r_{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 implicates of an n-variable function $f(x_1, \cdots, x_n)$ 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 $primes(f)$ 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 $\neg primes(\neg f)$. $primes(f)$ can also be viewed as a set and we define $\#lits(\rho)$ for $\rho \in primes(f)$ to be the number of literals in ρ.

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 $primes(f)$ 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 $\rho \in primes(f)$, that represents a solution to the following set covering problem [13]:

$$\begin{aligned} &\text{minimize} \sum_{\rho \,\in\, primes(f)} cost(\rho) \times z_\rho \\ &\text{subject to} \left(\bigvee_{\rho \,\in\, primes(f)} z_\rho \wedge \rho \right) = f \end{aligned} \tag{1}$$

where $cost(\rho) \triangleq \#lits(\rho)$. 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 $primes(f)$ 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 μ and column ρ indicates that minterm μ is (resp. is not) covered by prime implicant ρ. 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 $\mathcal{P}$ defined over a single unbounded sort $\texttt{node} \triangleq \{n_0, n_1, n_2, \cdots\}$ along with a) a finite set of *relations*[3] on $\texttt{node}$ that serve as $\mathcal{P}$'s state variables, and b) a finite set of *actions* that capture $\mathcal{P}$'s state transitions. The elements of $\texttt{node}$ are referred to as its *constants* and are assumed to be indistinguishable; they can be arbitrarily permuted without changing $\mathcal{P}$'s behavior.

A predicate Ψ on $\mathcal{P}$'s state variables is a closed quantified FOL expression. In prenex normal form (PNF) it can be expressed as $\Psi \triangleq Q_1 X_1 Q_2 X_2 \cdots Q_n X_n .\, \psi(X_1, X_2, \cdots, X_n)$ where $Q_i \in \{\forall, \exists\}$, $X_i \in \texttt{node}$ and ψ is a quantifier-free Boolean formula over $\mathcal{P}$'s relations. Following standard practice, we define $prefix(\Psi)$ as the string of quantifiers and bound variables, and refer to ψ as $matrix(\Psi)$. In the context of minimization, we further define the quantified cost of Ψ as

$$qCost(\Psi) = \#Q(prefix(\Psi)) + \#lits(matrix(\Psi)) \tag{2}$$

where $\#Q$ is the number of Ψ's quantified variables.

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

$$\left| vars^k \right| = \sum_{h \,\in\, relations} k^{arity(h)} \tag{3}$$

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

Partial Protocol Spec $\mathcal{P}$	Domain: $\texttt{node} \triangleq \{n_0, n_1, n_2, \cdots\}$ Relations: $a : \texttt{node} \mapsto \{0,1\}$, $b : \texttt{node} \mapsto \{0,1\}$		
Finite Protocol Instance $\mathcal{P}^3$	$\texttt{node}^3 \triangleq \{n_0, n_1, n_2\}$ $vars^3 = \{a(n_0), a(n_1), a(n_2), b(n_0), b(n_1), b(n_2)\}$ $Sym(\texttt{node}^3)$: $\{(\,), (n_0 n_1), (n_0 n_2), (n_1 n_2), (n_0 n_1 n_2), (n_0 n_2 n_1)\}$		
Explicit Clause Orbit	$(a(n_0) \vee b(n_1)) \wedge (a(n_0) \vee b(n_2)) \wedge$ $(a(n_1) \vee b(n_0)) \wedge (a(n_1) \vee b(n_2)) \wedge$ $(a(n_2) \vee b(n_0)) \wedge (a(n_2) \vee b(n_1))$	$(b(n_0) \vee b(n_1) \vee b(n_2))$	$(a(n_0) \vee b(n_0) \vee b(n_1) \vee b(n_2)) \wedge$ $(a(n_1) \vee b(n_0) \vee b(n_1) \vee b(n_2)) \wedge$ $(a(n_2) \vee b(n_0) \vee b(n_1) \vee b(n_2))$
Implicitly-Quantified Orbit	$\forall^3 N, M : (N = M) \vee a(N) \vee b(M)$ $qCost = 2 + 3 = 5$	$\exists^3 N : b(N)$ $qCost = 1 + 1 = 2$	$\forall^3 N, \exists^3 M : a(N) \vee b(M)$ $qCost = 2 + 2 = 4$
	(a) $\forall^3$ Quantification	(b) $\exists^3$ Quantification	(c) Mixed $\forall^3\exists^3$ Quantification

on a k-element set. In particular, $Sym(\texttt{node}^k)$ 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

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

where $\pi(f)$ is the result of applying the permutation π 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 $\texttt{node}^3$. 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 ($x \wedge x = x, x \vee x = x$) and commutativity ($x \wedge y = y \wedge x, x \vee y = y \vee x$).

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 $\texttt{node}^3$ 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 $\texttt{node}^3$ yields the exact set of clauses in the corresponding explicit orbits, except possibly for potential duplicates and tautologies. The correspondence between an explicit orbit $orbit_i^k$ and its finitely-quantified encoding Ψ_i^k can be expressed by a pair of related functions as

$$\begin{aligned} \Psi_i^k &= qInf(orbit_i^k) \\ orbit_i^k &= qIns(\Psi_i^k) \end{aligned} \quad (5)$$

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:

$$\begin{aligned} & r_{min}(\text{TC}) = \bigwedge_{1 \leqslant i \leqslant 8} \Psi_i \\ & \Psi_1 = \forall R.(a(R) \to \neg w(R)) \\ & \Psi_2 = \forall R.(a(R) \to \neg p(R)) \\ & \Psi_3 = \forall R.(a(R) \to \neg c(R)) \\ & \Psi_4 = \forall R.(p(R) \to \neg w(R)) \\ & \Psi_5 = \forall R.(c(R) \to \neg p(R)) \\ & \Psi_6 = \forall R.(c(R) \to \neg w(R)) \\ & \Psi_7 = \forall R.(w(R) \vee p(R) \vee c(R) \vee a(R)) \\ & \Psi_8 = \forall R_1, R_2. \\ & \quad c(R_1) \wedge \neg c(R_2) \wedge \neg p(R_2) \to (R_1 = R_2) \end{aligned} \quad (6)$$

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 $r_{min}(\text{TC}) \to S$ 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 $A_{min}(\text{TC}, S)$ it can be found using a minimal unsatisfiable subset (MUS) extractor, such as MARCO [23], from the UNSAT CNF formula

$$A_{min}(\text{TC}, S) = MUS[(r_{min}(\text{TC}) \wedge S) \wedge T \wedge \neg({r'}_{min}(\text{TC}) \wedge S')]$$

where T is the transition relation, the primes indicate a variable's next state, and the clauses of $r_{min}(\text{TC})$ 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,

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

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

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

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 $\mathcal{P}^k$. To achieve this, it takes advantage of two features of these formulas. The first, obvious, feature is the structural symmetry of $\mathcal{P}^k$. *QSM* preserves this symmetry by operating on prime implicant *orbits* rather than on individual prime implicants. The second, less obvious, feature is that the number of $\mathcal{P}^k$'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

$$\mathcal{P}^k \xrightarrow{QSM} r^k_{\text{min}} = \bigwedge_{1 \leqslant i \leqslant l} \Psi^k_i \tag{7}$$

In other words, *QSM* produces a minimum-cost conjunction of l finitely-quantified FOL formulas where each Ψ^k_i captures an orbit of r^k's prime implicates.

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_{ω^k}, for $\omega^k \in orbs(primes(\neg r^k))$, that represents a solution of the set covering problem

$$\begin{aligned} \min & \sum_{\omega^k \, \in \, orbs(primes(\neg r^k))} qCost(qInf(\neg\omega^k)) \times z_{\omega^k} \\ \text{s.t.} & \left(\bigvee_{\omega^k \, \in \, orbs(primes(\neg r^k))} z_{\omega^k} \wedge \omega^k \right) = \neg r^k \end{aligned} \tag{8}$$

Viewed as a formula, each such orbit ω^k is a disjunction of symmetric prime implicants; 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.

Algorithm 1 Symmetry-Aware Enumeration of PI Orbits

```
procedure EnumeratePIOrbits(¬r^k)
    ¬r^k_D ← dualRail(¬r^k)
    i ← 1, m ← |vars^k|, primeOrbits ← ∅
    while i ≤ m do
        (found, ρ_D) ← SAT?[¬r^k_D ∧ Σ_{1⩽j⩽m} (x^p_j + x^n_j) ⩽ i]
        if found then
            ω ← orbit^k(singleRail(ρ_D))
            primeOrbits ← primeOrbits ∪ {ω}
            ¬r^k_D ← ¬r^k_D ⋀_{ρ ∈ ω} dualRail(¬ρ)
        else
            i ← i + 1
    return primeOrbits
```

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

x^p	x^n	x
0	0	d
0	1	0
1	0	1
1	1	invalid

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 $(\neg x^p \vee \neg x^n)$ 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) 2^{m-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]:

$$\sum_{1 \leqslant j \leqslant m} (x^p_j + x^n_j) \leqslant l \tag{9}$$

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 ρ_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., $orbs(primes(\neg r^k)) = \{\omega_1^k, \cdots, \omega_n^k\}$, and let $[n] \triangleq \{1, 2, \cdots, n\}$. The covering constraints can now be captured by the CNF formula

$$\varphi^k \triangleq \bigwedge_{i \in [n]} (\neg z_i \vee \neg \omega_i^k) \tag{10}$$

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

$$SAT?[\varphi^k, \text{assume } chosenLiterals(\varphi^k)] \tag{11}$$

checks the satisfiability of φ^k assuming that all literals in $chosenLiterals(\varphi^k)$ 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

$$Z(sel) \triangleq (\bigwedge_{i \in sel} z_i) \wedge (\bigwedge_{i \notin sel} \neg z_i) \tag{12}$$

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

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

***Identifying Essential Orbits**:* A pending prime orbit ω_i^k 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

$$\begin{aligned} isEssential(\omega_i^k) &\triangleq SAT?[\neg(\omega_i^k \to \bigvee_{j \in sol \cup pnd \setminus \{i\}} \omega_j^k)] \\ &= SAT?[\omega_i^k \wedge \bigwedge_{j \in sol \cup pnd \setminus \{i\}} \neg \omega_j^k] \end{aligned}$$

Since ω_i^k 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 $\rho \in \omega_i^k$ to conclude if the whole orbit is or is not essential. This allows the above query to be re-expressed as

$$isEssential(\omega_i^k) = SAT?[\rho(\omega_i^k) \wedge \bigwedge_{j \in sol \cup pnd \setminus \{i\}} \neg \omega_j^k]$$

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

$$\boxed{\begin{aligned} &isEssential(\omega_i^k) = \\ &\quad SAT?[\varphi^k, \text{assume } \rho(\omega_i^k) \wedge Z(sol \cup pnd \setminus \{i\})] \end{aligned}} \tag{13}$$

***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:

$$coverage(\omega_i^k) \triangleq \#SAT?[\neg(\omega_i^k \to \bigvee_{j \in sol} \omega_j^k)]$$

Exact coverage can, thus, be expressed as

$$\boxed{\begin{aligned} &coverage(\omega_i^k) = \\ &\quad \#SAT?[\varphi^k, \text{assume } \rho(\omega_i^k) \wedge Z(sol)] \end{aligned}} \tag{14}$$

Coverage is used to remove completely covered orbits from pnd (when $coverage = 0$) and to rank partially-covered orbits for the branching step (when $coverage > 0$). 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 cov.

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

At each invocation, *QSM* performs the following steps:

- Line 2: It updates the current covering requirements (encoded by pnd, sol, and 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 *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          else
8              return NOSOLUTION
9      LB ← qCost(sol)
10     if LB ≥ UB then
11         return NOSOLUTION
12     i ← chooseOrbit(pnd, cov)
13     S^{with_i} ← QSM(pnd \ {i}, sol ∪ {i}, cov, UB)
14     if qCost(S^{with_i}) = LB then
15         return (S^{with_i})
16     S^{without_i} ← QSM(pnd \ {i}, sol, cov, UB)
17     return BESTSOLUTION(S^{with_i}, S^{without_i})

18 procedure reduce(pnd, sol, cov)
19     (existEss, pnd, sol) ← addEssentials(pnd, sol)
20     (existCov, pnd, cov) ← removeCovered(pnd, sol, cov)
21     if existEss ∨ existCov then
22         (pnd, sol, cov) ← reduce(pnd, sol, cov)
23     return (pnd, sol, cov)

24 procedure addEssentials(pnd, sol)
25     essentials ← ∅
26     for each orbit ∈ pnd do
27         if [isEssential(orbit, pnd, sol)] then
28             essentials ← essentials ∪ {orbit}
29     sol ← sol ∪ essentials
30     pnd ← pnd \ essentials
31     return (|essentials| > 0, pnd, sol)

32 procedure removeCovered(pnd, sol, cov)
33     covered ← ∅
34     for each orbit ∈ pnd do
35         [cov[orbit] ← coverage(orbit, sol)]
36         if cov[orbit] = 0 then
37             covered ← covered ∪ {orbit}
38     pnd ← pnd \ covered
39     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.

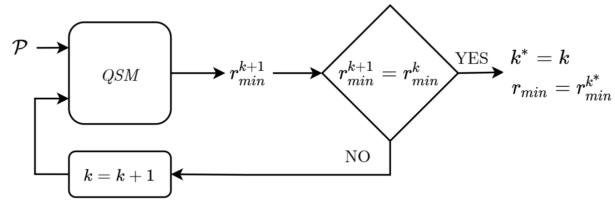

Fig. 1: *QSM* Syntactic Fixed Point Convergence

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[4] $r_{min}^1, r_{min}^2, \cdots$. An interesting empirical observation is that this sequence reaches a *syntactic* fixed point (Figure 1) at some value $k^\star$ defined as:

$$\begin{aligned}
&\bullet\ r_{\min}^{k^\star} = \bigwedge_{1 \leqslant i \leqslant l} \Psi_i^{k^\star} \\
&\bullet\ r_{\min}^{k^\star+1} = \bigwedge_{1 \leqslant i \leqslant l} \Psi_i^{k^\star+1} \\
&\bullet\ \forall i \in [1,l] : \mathit{prefix}(\Psi_i^{k^\star+1}) = \mathit{prefix}(\Psi_i^{k^\star}) \\
&\bullet\ \forall i \in [1,l] : \mathit{matrix}(\Psi_i^{k^\star+1}) = \mathit{matrix}(\Psi_i^{k^\star})
\end{aligned} \tag{15}$$

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 $k^\star$ become redundant and do not introduce new behaviors beyond $k^\star$. 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

$$r_{min} = \bigwedge_{1 \leqslant i \leqslant l} \Psi_i \tag{16}$$

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

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

TABLE II: *QSM* Experimental Results†

Protocol	Memory MB	CPU Time, sec					Number of				r_{min}	Assertions	
		Total	BDD	PI Gen	qInf	Cov	vars	r^k cubes	$\neg r^k$PIs	Orbits	Orbits	Human	IC3PO
tla-consensus	59	9	4	0	4	0*	3	3	3	1	1	0	0
tla-tcommit	67	9	4	0	4	0	8	12	26	13	8	2	1
tla-twophase	67	7197	4	1	4	7188	17	29	252	134	?	11	11
distai-ricart-agrawala	67	9	4	0	4	0*	10	4	10	6	4	2	2
i4-lock-server	67	17	8	1	8	1*	8	16	14	3	3	1	1
pyv-sharded-kv	104	58	22	10	17	10*	42	324	420	15	8	4	5
pyv-sharded-kv-no-lost-keys	106	68	22	15	18	14	42	324	422	16	9	1	1
ex-simple-decentralized-lock	67	18	8	1	8	1*	24	80	198	18	18	4	3
pyv-firewall	67	56	6	1	7	42	15	23	348	62	5	1	2
pyv-lockserv	67	9	4	0	4	0*	13	10	46	13	13	8	8
ex-lockserv-automaton	67	9	4	0	4	0*	13	10	46	13	13	1	8
ex-toy-consensus	108	29	14	2	12	2	27	138	510	19	4	2	2
ex-naive-consensus	67	21	9	2	8	2*	27	189	396	14	3	3	3
ex-simple-election	47056	3812	3773	2	11	27	27	220	849	55	7	2	3
pyv-toy-consensus-forall	67	31	13	3	13	3	23	1072	518	19	6	3	3
pyv-toy-consensus-epr	48912	3622	3607	1	11	3	24	94	534	35	6	3	3
pyv-consensus-epr	913	7401	4675	74	32	2621	72	1318	19818	743	16	6	5

† 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, PIs, and Orbits are at the cutoff size.
*For these protocols, r_{min} was found without any branch-and-bound search.

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^k_{min}.

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 $\mathcal{P}^k$. 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.

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[5], for r^k. The excessive run time of the covering step in tla-twophase and pyv-consensus-epr was a direct consequence of the large number of PIs 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

[5]The current BDD front-end limited the set of protocols our prototype can support.

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

Protocol	Finite instance sizes † ‡
tla-consensus	$\texttt{value} = 2 \mapsto 3$
tla-tcommit	$\texttt{resource-manager} = 2$
tla-twophase	$\texttt{resource-manager} = 2$
distai-ricart-agrawala	$\texttt{node} = 2$
i4-lock-server	$\texttt{client} = 2 \mapsto 3,\ \texttt{server} = 1 \mapsto 2$
pyv-sharded-kv	$\texttt{key} = 2,\ \texttt{node} = 2 \mapsto 3,\ \texttt{value} = 2 \mapsto 3$
pyv-sharded-kv-no-lost-keys	$\texttt{key} = 2,\ \texttt{node} = 2 \mapsto 3,\ \texttt{value} = 2 \mapsto 3$
ex-simple-decentralized-lock	$\texttt{node} = 2 \mapsto 4$
pyv-firewall	$\texttt{node} = 2 \mapsto 3$
pyv-lockserv	$\texttt{node} = 2 \mapsto 3$
ex-lockserv-automaton	$\texttt{node} = 2 \mapsto 3$
ex-toy-consensus	$\texttt{node} = 2 \mapsto 4,\ \texttt{quorum} = 1 \mapsto 4,\ \texttt{value} = 2 \mapsto 3$
ex-naive-consensus	$\texttt{node} = 3 \mapsto 4,\ \texttt{quorum} = 3 \mapsto 4,\ \texttt{value} = 3$
ex-simple-election	$\texttt{acceptor} = 2 \mapsto 3,\ \texttt{proposer} = 2 \mapsto 3,\ \texttt{quorum} = 1 \mapsto 3$
pyv-toy-consensus-forall	$\texttt{node} = 2 \mapsto 4,\ \texttt{quorum} = 1 \mapsto 4,\ \texttt{value} = 2 \mapsto 3$
pyv-toy-consensus-epr	$\texttt{node} = 2 \mapsto 3,\ \texttt{quorum} = 1 \mapsto 3,\ \texttt{value} = 2 \mapsto 3$
pyv-consensus-epr	$\texttt{node} = 2 \mapsto 4,\ \texttt{quorum} = 1 \mapsto 4,\ \texttt{value} = 2 \mapsto 3$

† $\texttt{s} = x$ denotes sort $\texttt{s}$ has both initial size and final cutoff size x
‡ $\texttt{s} = x \mapsto y$ denotes sort $\texttt{s}$ 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 paraVerifier [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

Sub-orbits	invariant [pi19] forall N1, N2, N3.	(∼has_lock(N1) | ∼message(N3, N2) | (N1 = N2) | (N1 = N3) | (N2 = N3))
	invariant [pi25] forall N1, N2.	(∼has_lock(N1) | ∼message(N2, N2) | (N1 = N2))
	invariant [pi31] forall N1, N2.	(∼has_lock(N1) | ∼message(N1, N2) | (N1 = N2))
	invariant [pi37] forall N1, N2.	(∼has_lock(N1) | ∼message(N2, N1) | (N1 = N2))
	invariant [pi43] forall N1.	(∼has_lock(N1) | ∼message(N1, N1))
Equivalent Super Orbit	invariant [pi19_pi43] forall N, S, D.	(∼has_lock(N) | ∼message(S, D))

An important extension would be to derive r_{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_{min} that can serve as minimum strengthening assertions for given safety properties.

The experimental results strongly hint that the r_{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_{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

[1] C. Barrett and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Model Checking*. Springer, 2018, pp. 305–343.

[2] C. Barrett, P. Fontaine, and C. Tinelli, "The Satisfiability Modulo Theories Library (SMT-LIB)," www.SMT-LIB.org, 2016.

[3] A. Karbyshev, N. Bjørner, S. Itzhaky, N. Rinetzky, 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

[4] H. Ma, A. Goel, J.-B. Jeannin, M. Kapritsos, B. Kasikci, and K. A. Sakallah, "I4: Incremental Inference of Inductive Invariants forVerification of Distributed Protocols," in *The 27th ACM Symposium on Operating Systems Principles (SOSP 2019)*, Huntsville, Ontario, Canada, October 2019, pp. 370–384.

[5] 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

[6] A. Goel and K. A. Sakallah, "On Symmetry and Quantification: A New Approach to Verify Distributed Protocols," in *13th Annual NASA Formal Methods Symposium (NFM 2021)*, Langley, Virginia, May 2021, pp. 131–150. [Online]. Available: https://arxiv.org/abs/2103.14831

[7] 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

[8] 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.

[9] J. Yao, R. Tao, R. Gu, and J. Nieh, "{DuoAI}: Fast, automated inference of inductive invariants for verifying distributed protocols," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 485–501.

[10] N. Eén and N. Sörensson, "An extensible sat-solver," in *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, ser. Lecture Notes in Computer Science, E. Giunchiglia and A. Tacchella, Eds., vol. 2919. Springer, 2003, pp. 502–518. [Online]. Available: https://doi.org/10.1007/978-3-540-24605-3_37

[11] A. Biere, M. Heule, and H. van Maaren, *Handbook of satisfiability*. IOS press, 2009, vol. 185.

[12] J. B. Fraleigh, *A First Course in Abstract Algebra*, 6th ed. Reading, Massachusetts: Addison Wesley Longman, 2000.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[14] W. V. Quine, "The problem of simplifying truth functions," *The American mathematical monthly*, vol. 59, no. 8, pp. 521–531, 1952.

[15] ——, "A way to simplify truth functions," *The American mathematical monthly*, vol. 62, no. 9, pp. 627–631, 1955.

[16] E. J. McCluskey, "Detection of group invariance or total symmetry of a boolean function," *The Bell System technical journal*, vol. 35, no. 6, pp. 1445–1453, 1956.

[17] L. Lamport, "The temporal logic of actions," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 16, no. 3, pp. 872–923, 1994.

[18] 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.

[19] P. Wolper, "Expressing interesting properties of programs in propositional temporal logic," in *Proceedings of the 13th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, 1986, pp. 184–193.

[20] 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

[21] "A TLA+ specification of the Transaction Commit protocol," https://github.com/tlaplus/Examples/blob/master/specifications/transaction_commit/TCommit.tla.

[22] J. Gray and L. Lamport, "Consensus on transaction commit," *ACM Transactions on Database Systems (TODS)*, vol. 31, no. 1, pp. 133–160, 2006.

[23] M. H. Liffiton, A. Previti, A. Malik, and J. Marques-Silva, "Fast, flexible mus enumeration," *Constraints*, vol. 21, no. 2, pp. 223–250, 2016.

[24] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, "Symbolic Model Checking: 10^{20} States and Beyond," in *Proceedings. Fifth Annual IEEE Symposium on Logic in Computer Science*, 1990, pp. 428–439.

[25] V. M. Manquinho, P. F. Flores, J. P. M. Silva, and A. L. Oliveira, "Prime implicant computation using satisfiability algorithms," in *9th International Conference on Tools with Artificial Intelligence, ICTAI '97, Newport Beach, CA, USA, November 3-8, 1997*. IEEE Computer Society, 1997, pp. 232–239. [Online]. Available: https://doi.org/10.1109/TAI.1997.632261

[26] S. Jabbour, J. Marques-Silva, L. Sais, and Y. Salhi, "Enumerating prime implicants of propositional formulae in conjunctive normal form," in *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, ser. Lecture Notes in Computer Science, E. Fermé and J. Leite, Eds., vol. 8761. Springer, 2014, pp. 152–165. [Online]. Available: https://doi.org/10.1007/978-3-319-11558-0_11

[27] R. Martins, S. Joshi, V. M. Manquinho, and I. Lynce, "Incremental cardinality constraints for maxsat," in *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, ser. Lecture Notes in Computer Science, B. O'Sullivan, Ed., vol. 8656. Springer, 2014, pp. 531–548. [Online]. Available: https://doi.org/10.1007/978-3-319-10428-7_39

[28] R. Martins, S. Joshi, V. Manquinho, and I. Lynce, "Incremental cardinality constraints for maxsat," in *Principles and Practice of Constraint Programming: 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings 20*. Springer, 2014, pp. 531–548.

[29] G. D. Hachtel and F. Somenzi, *Logic synthesis and verification algorithms*. Springer Science & Business Media, 2007.

[30] C. P. Gomes, A. Sabharwal, and B. Selman, "Model counting," in *Handbook of satisfiability*. IOS press, 2021, pp. 993–1014.

[31] 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.

[32] 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.

[33] "A collection of distributed protocol verification problems," https://github.com/aman-goel/ivybench.

[34] 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.

[35] 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.

[36] I. Berkovits, M. Lazić, G. Losa, O. Padon, and S. Shoham, "Verification of threshold-based distributed algorithms by decomposition to decidable logics," in *International Conference on Computer Aided Verification*. Springer, 2019, pp. 245–266.

[37] A. Goel and K. A. Sakallah, "IC3PO: IC3 for Proving Protocol Properties," https://github.com/aman-goel/ic3po.

[38] 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.

[39] S. Owre, J. M. Rushby, and N. Shankar, "Pvs: A prototype verification system," in *International Conference on Automated Deduction*. Springer, 1992, pp. 748–752.

[40] 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.

[41] 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

[42] 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.

[43] K. v. Gleissenthall, R. G. Kıcı, A. Bakst, D. Stefan, and R. Jhala, "Pretend synchrony: synchronous verification of asynchronous distributed programs," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–30, 2019.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] M. J. Ghazala, "Irredundant disjunctive and conjunctive forms of a boolean function," *IBM J. Res. Dev.*, vol. 1, no. 2, pp. 171–176, 1957. [Online]. Available: https://doi.org/10.1147/rd.12.0171

[49] A. Goel and K. A. Sakallah, "Regularity and Quantification: A New Approach to Verify Distributed Protocols," *Innovations in Systems and Software Engineering (ISSE)*, pp. 1–19, September 2022.

[50] ——, "Towards an Automatic Proof of Lamport's Paxos," in *Formal Methods in Computer-Aided Design (FMCAD)*, R. Piskac and M. W. Whalen, Eds., New Haven, Connecticut, October 2021, pp. 112–122.

[51] A. Goel, S. Merz, and K. A. Sakallah, "Towards an automatic proof of the bakery algorithm," in *Formal Techniques for Distributed Objects, Components, and Systems*, M. Huisman and A. Ravara, Eds. Cham: Springer Nature Switzerland, 2023, pp. 21–28.