First-Order Quantified Separators

Jason R. Koenig
Stanford University
USA
jrkoenig@cs.stanford.edu

Oded Padon
Stanford University
USA
padon@cs.stanford.edu

Neil Immerman
University of Massachusetts Amherst
USA
immerman@cs.umass.edu

Alex Aiken
Stanford University
USA
aiken@cs.stanford.edu

Abstract

Quantified first-order formulas, often with quantifier alternations, are increasingly used in the verification of complex systems. While automated theorem provers for first-order logic are becoming more robust, invariant inference tools that handle quantifiers are currently restricted to purely universal formulas. We define and analyze first-order quantified separators and their application to inferring quantified invariants with alternations. A *separator* for a given set of positively and negatively labeled structures is a formula that is true on positive structures and false on negative structures. We investigate the problem of finding a separator from the class of formulas in prenex normal form with a bounded number of quantifiers and show this problem is NP-complete by reduction to and from SAT. We also give a practical separation algorithm, which we use to demonstrate the first invariant inference procedure able to infer invariants with quantifier alternations.

CCS Concepts: • **Theory of computation** → **Complexity theory and logic**; • **Software and its engineering** → *Formal methods.*

Keywords: invariant inference, first-order logic

ACM Reference Format:
Jason R. Koenig, Oded Padon, Neil Immerman, and Alex Aiken. 2020. First-Order Quantified Separators. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '20), June 15–20, 2020, London, UK*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3385412.3386018

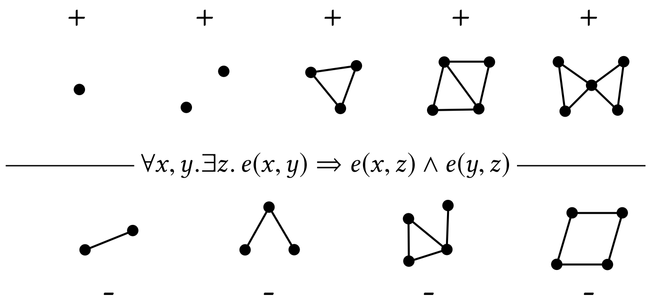
+
+
+
+
+
$\forall x, y. \exists z.\, e(x, y) \Rightarrow e(x, z) \land e(y, z)$
-
-
-
-

Figure 1. A set of labeled structures represented by graphs, and a quantified separator between them. The separator may be interpreted as "every edge is part of a triangle." The structures are defined over a signature with a single sort for vertices and a single symmetric relation $e(\cdot, \cdot)$ for edges.

1 Introduction

Considerable human effort is currently required to verify software systems where quantified formulas are necessary to express invariants. A few systems have support for inferring universally quantified invariants, but there are no general approaches to inferring quantified invariants with even one quantifier alternation (i.e. a formula with nested ∀ and ∃), even though there are domains in which such invariants are needed [24]; currently such invariants must be discovered by hand.

It is well-known that program invariants are *separators* of states, evaluating to *true* on the good (reachable) states and *false* on the bad (error) states of a system. A common approach in invariant inference algorithms for quantifier-free invariants is to learn a formula that separates good and bad states (e.g., [26]); with enough of the right sort of examples, the discovered separator will hopefully be an invariant.

In this paper we introduce the problem of *first-order quantified separability*, which we believe is a key step towards improved automation for verification problems requiring quantified invariants. We give algorithms and complexity results for inferring separators that are quantified first-order formulas. We give experimental results that show our algorithm is practical for learning formulas taken from a corpus of invariants of distributed protocols. We further show how to adapt IC3/PDR [4] to use separators, and demonstrate the first system able to infer invariants with alternations for challenging distributed protocols.

The separability problem is to compute a separator if it exists, or report that there is no separator. An example of a separation query and a solution is given in Figure 1. We consider separators expressed in classical first-order logic. The separability problem has some connections to graph isomorphism [17], Boolean function learning [5], and interpolation [21]. If arbitrary first-order formulas are permitted, then it becomes very easy to separate any two finite sets of distinct labeled structures, because it is possible to construct a formula that is true for exactly one structure. Thus, the disjunction of such formulas for each positive structure (or the negation of a disjunction of such formulas for each negative structure) will be a separator. However, such separators obviously overfit to the examples and fail to generalize, and the ability to find separators that generalize to examples of positive and negative structures not previously seen is clearly important for application to invariant inference.

Thus, we need restrictions on the first-order formulas we will consider. We focus on restricting the structure of the quantifiers, including bounding the maximum number of distinct quantified variables, bounding the depth of quantification, restricting to purely universal formulas, and especially restricting to *prenex normal form*, i.e. formulas where all quantifiers are at the beginning. It is well-known that any first-order formula can be efficiently transformed into an equivalent formula with the same number of quantifiers in prenex form. We will see that prenex form is both expressive and not prone to overfitting. We propose formulas with at most k quantifiers in prenex normal form (k-prenex formulas) as an interesting class of separators. In particular, the division into a prefix quantifier structure and a quantifier-free body (the *matrix*) enables algorithms that can directly reuse existing SAT solvers.

We show that some other separability problems, such as quantifier-depth k formulas, are in P. We show that for fixed $k \geq 2$, the k-prenex separability problem is NP-complete. The proof reduces a SAT problem to a carefully constructed set of highly symmetric structures that force the separator to encode an assignment to the SAT problem in the matrix (Boolean) part of the formula. By using the two player game semantics of first-order logic, we show that the separability of this set of structures is insensitive to permutations of the quantifiers, and depends only on whether the formula has the right *number* of $\forall$ and $\exists$ quantifiers.

In the other direction, we give an algorithm for reducing k-prenex separability to SAT, completing the proof of NP-completeness. We develop the connection to SAT by showing that the quantifier structure can be eagerly computed up front, leaving only a SAT problem that corresponds to a search for the Boolean part of the formula. While this algorithm is sufficient for solving the decision problem of separability, it does not in general give good formulas for practical separation problems. In particular, for a given set of structures it may not give us the minimum k that allows them to be separated. We show how to find a separating formula with the minimum number of quantifiers k. We also give an optimization algorithm to minimize syntactic measures of the quantifier-free part of this formula.

Contributions. The contributions of this work are:

1. The identification of k-prenex separation as a natural and useful problem (Section 3),
2. A proof of NP-completeness of k-prenex separation (Section 4),
3. An algorithm for solving k-prenex separation along with an evaluation of its performance and scalability on human-authored formulas (Sections 5 and 7),
4. Preliminary evidence that separators can be used to find invariants by adapting IC3/PDR to use separators and the first demonstration of an approach able to infer invariants with quantifier alternations (Sections 6 and 7).

2 Background

We give some background on first-order logic (FOL) as used in this work, and we discuss prior work.

2.1 First-Order Logic

In this work we use first-order, many-sorted logic with equality. Formulas in this logic are defined relative to some signature naming the constant, relation, and function symbols along with their sorts. We consider only finite signatures, i.e. ones with a finite number of sorts and symbols. *Terms* are constants, variables, or function symbols applied to other terms. Examples include x, $f(y)$, and $g(x, f(z))$. *Atoms* are a relation symbol or equality applied to terms of appropriate sorts, and *literals* are atoms or their negation. Examples of atoms include $p(x)$, $x = y$, and $r(x, f(y))$. Finally, *formulas* are the closure of literals under conjunction, disjunction and quantification. An example formula over a signature of two sorts s_1 and s_2 is $\forall x : s_1.\, \exists y : s_2.\, (p(x) \wedge \neg r(x, y)) \vee x = f(y)$. A standard result in logic is that any formula is logically equivalent to one in *prenex normal form*, in which all quantifiers are lifted to the front of the formula. For such *prenex* formulas, the quantifier part is called the *prefix* and the remaining Boolean structure is called the *matrix* (which we usually denote φ).

A *structure* M over some signature S, sometimes called a *model*, is a set of sorted elements, along with an interpretation for each constant, relation and function symbol of S. We only consider *finite structures*, i.e. ones in which the set of elements is finite. An *assignment* is a mapping, σ, from variables to elements of M. A structure M may be augmented with σ to form $M \cup \sigma$ by adding the variables as new constants interpreted according to σ. We say that a structure M *satisfies* a formula p, written $M \models p$, if p is true in M.

2.2 Game Semantics of Logic

The standard semantics for first-order logic may be defined via a two player game, with one player as $\forall$ and one as $\exists$ [15]. We use a semantics simplified to answer $M \models p$ for prenex formula p with matrix φ: the two players take turns picking appropriately sorted elements according to their order in the prefix, with the game ending in some assignment σ when the prefix is exhausted. The $\exists$ player wins if and only $M \cup \sigma \models \varphi$. The semantics becomes: $M \models p$ if and only if $\exists$ has a winning strategy; otherwise $\forall$ has a winning strategy and we say $M \not\models p$. The best-known algorithms for deciding $M \models p$ are exponential in the number of quantifiers of p, as in the worst case they must explore all the exponentially many assignments of quantified variables.

2.3 Quantifier-Free Types

Intuitively, two tuples $a_1, \ldots, a_k \in M$, $b_1, \ldots, b_k \in N$ of elements from two structures over the same signature have the same *quantifier-free type*, or *QF-type*, if they cannot be distinguished by a quantifier-free formula. Formally, the quantifier-free type of $a_1, \ldots, a_k \in M$ is the set of quantifier-free formulas with free variables $x_1, \ldots, x_k$ that are satisfied by M and the assignment $[a_1/x_1, \ldots, a_k/x_k]$. We also define the b*-bounded* QF-type to only allow formulas in which all function symbols are nested at most b levels deep. In our evaluation (Section 7) we fix $b = 1$.

2.4 Prior Work

Our work draws on ideas from logic and machine learning, and is motivated by applications of quantified formulas in verification.

2.4.1 Logical Separability. Distinguishability of two structures with a quantified formula has been investigated through the study of Ehrenfeucht-Fraïssé (EF) games [16]. These games characterize the structures that may be separated by a formula with quantifier rank k, which is the maximal depth of a quantifier in the formula. EF games are traditionally only defined for a pair of structures, while we are interested in separating sets of structures. A related usage is graph isomorphism problems. Determining whether a pair of structures can be separated by *any* first-order formula is the same question as asking if they are isomorphic, which is a generalization of graph isomorphism.

2.4.2 Interpolants. Separators are related to the concept of a *interpolant*, which has been applied to software and hardware verification [21, 22]. A interpolant can be viewed as a separator between sets of states described by formulas. While interpolants separate symbolic sets of states and we consider finite, concrete sets of states, the two are connected because one can be used to implement the other (i.e., by creating a formula that encodes a finite set, or in the other direction by using counterexample guided refinement of the concrete sets).

Existing work finds quantified interpolants for alternation-free formulas [8]. Another existing interpolation technique [1] is similar to this work in that it reduces to SAT and minimizes a syntactic measure of complexity, but it considers quantifier-free interpolants over the theory of linear rational arithmetic.

2.4.3 Boolean Learnability. The problem of learning a Boolean function from examples can be seen as a separability problem where the formula is restricted to propositional logic. This problem is often investigated in models like PAC (*probably approximately correct*) learning from statistical machine learning theory, where the separator need only be mostly correct on the examples. In this work we are interested only in exact separators that label all examples correctly. For example, [5] shows that in the exact learning setting, only a polynomial number of examples are required if the true separator has both a short CNF and DNF representation. In contrast the worst case to learn an arbitrary Boolean function of n inputs requires 2^n examples, one for each input point. Recently, [10] showed that for some classes of Boolean functions, learning inductive invariants is harder than exact learning.

2.4.4 VC-dimension. The VC-dimension of a class of classifier functions $\mathcal{F}$ over some domain D is the size of the largest set $C \subseteq D$ that the class can *shatter*. A class $\mathcal{F}$ shatters a set C if for any labeling of elements of C as positive and negative, a function from $\mathcal{F}$ assigns that labeling. Stated another way, the VC-dimension is the largest set that can be separated by $\mathcal{F}$ in every possible way. While VC-dimension was originally defined for statistical learning [30], it has been applied to study exact learning for program analysis [28].

2.4.5 Quantified Formulas in Verification. Despite their computational expense and possible undecidability, quantified formulas are used in many verification tools. In Ivy [25], Alloy [18], and Dafny [20], quantified formulas including quantifier alternation are part of the user's interface to the system. Existing invariant inference techniques based on IC3/PDR [4, 9] such as PDR$^\forall$ [19] are restricted to universally quantified invariants, and systems that would be modeled naturally with existential quantifiers must be manually transformed (if possible) to eliminate existential quantifiers.

3 The Separability Problem

To define the separability problem, we need to fix a class of formulas to consider as separators. If we allow arbitrary first-order formulas then the problem is trivial: we can write down a formula that satisfies exactly a given structure and a disjunction of such formulas for the positive structures will separate any set provided no structure is both positive and negative. Such formulas have at least as many quantifiers as structure elements, so it is natural to limit the quantifiers in some way.

We focus on the separability problem restricted to prenex normal formulas with at most k quantifers (*k-prenex*):

Definition 3.1. k-SEP is the decision problem of determining whether a given set of positively and negatively labeled structures is separable by a prenex formula with at most k quantifiers.

In addition, we can define the separation problem for when we are given a particular prefix to separate with:

Definition 3.2. fixed-k-SEP is the decision problem of determining whether a given set of positively and negatively labeled structures is separable by a prenex formula with exactly a given prefix of size k.

For example, Figure 1 is separable in 3-SEP but not in 2-SEP. Similarly, it is separable in 3-fixed-SEP for the prefix $\forall\forall\exists$ but not for the prefix $\exists\exists\exists$.

In the remainder of this section, we will explore the properties of other classes of formulas, such as those with quantifiers nested up to depth k (*k-depth*). We will show how these properties relate to those of k-prenex, and explain why some theoretical properties indicate that k-prenex might be a good choice for separation. Sections 4–7 explore k-prenex separation, and do not depend on the material in the rest of this section.

3.1 k-Depth Separability is in P

The quantifier-depth of a first-order formula, p, is the maximum depth of nesting of quantifiers in p. It is easy to see from known results [17] that for each fixed k, k-depth separability is in P by computing the model-theoretic *types* of the structures. The C-type, for some class of formula C, is the set of all formulas from C true of that structure. If two structures have the same type, they are inseparable. If the types are different then there is a formula in one set and not the other that can be the separator.

Proposition 3.3. For a fixed relational signature, given a positively labeled and b negatively labeled structures where each is of maximum size n and all have total size s, testing if they are k-depth separable, and if so, finding a depth k separator, can be done in time $O(s + (a + b)n^k)$.

The k-dimensional Weisfeiler-Leman algorithm, known for its applications to graph isomorphism, computes a coloring of all k-tuples of vertices in an input graph. As [17] shows, this coloring of k-tuples is exactly the C^{k+1}-*type* of the k-tuple, where C^k is first-order logic with a fixed set of k variables $x_1, \dots x_k$ (which may be arbitrarily requantified) and *counting quantifiers*.

The time to compute this coloring is given as follows:

Proposition 3.4. [17] We can compute the C^k types of a given n-vertex graph in time $O(n^k \log n)$.

The algorithm in Prop. 3.4 works just as well, in the same running time, for L^k, first-order logic with at most k variables but no counting quantifiers. Furthermore, this algorithm works by incrementally computing the L^k or S^k type of quantifier-depth 0, 1, 2, . . ., until a fixed point is reached. Stopping after k rounds reduces the time needed to compute the depth k L^k type to just $O(n^k)$. In depth k, having at most k variables is no restriction. Furthermore, the same algorithm works for any relational structure with relations of arity at most k.

Corollary 3.5. Given a relational structure of size s, universe size n and maximal arity at most k, we can compute its depth k type in time $O(s + n^k)$.

Finally, to prove Prop. 3.3, we simultaneously compute the depth k types of the given structures. They are k-depth separable iff the set of types occurring in the positive structures is disjoint from the set of types occurring in the negative structures. In this case, a depth k separator is just a disjunction of the types of the positive structures. We can separate structures over signatures with function symbols as long as the nesting in the allowed separators is bounded, as fresh relations can be introduced to represent the finite set of atoms containing function symbols.[1]

These disjunctive k-depth separators encode the positive structures they separate directly, and intuitively we do not expect them to generalize well. In contrast with k-depth formulas, k-prenex formulas must share the k quantifiers amongst all the structures, so we might expect these separators, when they exist, to find some common property of the structures. One theoretical tool to analyze this intuitive overfitting behavior is by calculating the VC-dimension of these classes.

3.2 VC-Dimension of k-Depth is Exponentially Larger than k-Prenex

We analyze the VC dimension of k-depth and k-prenex formula classes over a fixed signature which has one binary relation r and no other symbols (besides equality).

k-Prenex. A bound on the VC-dimension of a class can be obtained based on the size of the class. Recall that to shatter a set of size n, we need at least 2^n different functions, so the VC-dimension is bounded above by the logarithm of the size

[1]For example, introduce fresh relations $r'_1(x, y) = r(f(x), y)$, $r'_2(x, y) = r(f(x), g(y))$, etc. and remove function symbols. When a separator exists, these relations can be expanded to obtain a separator of the original structures.

Table 1. Summary of important properties of separator classes. VC-dimension is for the signature with a single binary relation.

Class	Sep. Complexity	VC-dim	$\wedge/\vee$-closed?
full FOL	graph-iso.	∞	✓
alternation-free	graph-iso.	∞	✓
$\forall^*/\exists^*$	NP-complete	∞	
$\forall^k/\exists^k$	P[†]	$\leq 2^{2k^2}$	
k-depth	P[†]	$\geq 2^{\|G_k\|}$	✓
k-prenex	NP-complete[†]	$\leq k + 2^{2k^2}$	

[†]for fixed k and signature.

of the class. There are 2^k different prefixes, and $2k^2$ different atoms (k^2 for both the relation and equality). The number of matrices is then $2^{2^{2k^2}}$, and the overall number of formulas is $2^{k+2^{2k^2}}$. Thus the VC-dimension of k-prenex formulas is at most $k + 2^{2k^2}$.

***k*-Depth.** We show that k-depth formulas may shatter a set of structures which encode a large number of disjoint graphs. Let G_k be the set of all distinct unlabeled directed graphs without self-loops on $k-1$ vertices. For a graph $g \in G_k$, we construct a graph *gadget* g' as follows: Add a new vertex v_g with a self loop, $r(v_g, v_g)$ plus edges to all the vertices in g. This new vertex prevents formulas designed to match one gadget from spuriously matching a subgraph of another gadget. Now we build a set of structures S with all $2^{|G_k|}$ patterns of presence or absence of graph gadgets for graphs $g \in G$. For each g, we can construct a formula:

$$\exists x_0, x_1, \ldots, x_{k-1}.\ r(x_0, x_0) \wedge r(x_0, x_1) \wedge \cdots \wedge r(x_0, x_{k-1}) \\ \wedge (x_{i>0} \text{ are related as in } g)$$

This formula is true precisely when the gadget for g appears in the structure. To isolate a particular structure, we can conjunct the formulas for the present graphs with the negations of the formulas of all the absent ones. Then by taking the disjunction of these formulas for a set of structures, we can separate any subset of S. Thus the VC-dimension of k-depth formulas is at least $2^{|G_k|}$ which is exponentially larger than $k + 2^{2k^2}$ because $|G_k|$ is exponential in k [13]. These results can be extended to other signatures as long as they have at least one non-unary relation (to encode the graph gadgets) and the function nesting depth is fixed (as in Section 3.1).

3.3 Summary of Separator Classes

In addition to the classes of separators we have already looked at, we can consider a few more related classes:

1. Alternation-free: Formulas in which $\exists$ does not appear inside $\forall$ and vice versa.
2. $\forall^*/\exists^*$. A pair of classes, each consisting of an arbitrary number of quantifiers of one kind. These classes are closely related because swapping the labels on the structures negates the separator and switches $\forall \leftrightarrow \exists$.
3. $\forall^k/\exists^k$. A pair of classes, each with prefixes of at most k quantifiers of one kind. Each is a subclass of k-prenex formulas.

If a class of formulas is closed under $\wedge$ and $\vee$, a separator of that class exists iff every pair of positive and negative structures can be separated by possibly different formulas. Let Φ_{ij} separate positive structure i from negative structure j. Then the formula:

$$\bigvee_i \left(\bigwedge_j \Phi_{ij} \right)$$

is true for positive structures and false for negative structures. Thus separator classes closed under $\wedge/\vee$ are actually not desirable, because they allow a separator to be constructed from pairwise separators.

The pairwise argument allows us to give the complexity of full FOL and alternation-free separators as equivalent to graph isomorphism, because a *pair* of structures are separable with these classes iff they are not isomorphic. Isomorphism of first-order structures is equivalent to graph isomorphism, which is in NP but not known to be NP-complete or in P. The complexity of $\forall^*/\exists^*$ is equivalent to subgraph isomorphism because the structures are separable iff no negative structure is a substructure of a positive structure. Subgraph isomorphism is known to be NP-complete [6].

To summarize, for each of these classes we give the complexity of the separability decision problem, the VC-dimension, and whether it is closed under $\vee$ and $\wedge$ in Table 1. If we adopt the common hypothesis that low VC-dimension helps avoid overfitting, then desirable candidates are $\forall^k/\exists^k$ and k-prenex. While $\forall^k/\exists^k$ has desirable properties including low computational complexity, purely universal or existential formulas cannot express invariants of many systems. We thus propose k-prenex formulas as a separator class k-SEP, because it is expressive and our evaluation will show that it is often tractable in practice. One interesting property of the complexity of these classes is that the complexity of k-prenex is higher than either k-depth or $\forall^k/\exists^k$, while in terms of inclusion it sits between the two.

We now show that k-prenex separation (for fixed k) is NP-complete, by first showing it is NP-hard and then presenting an algorithm to reduce k-prenex separation to SAT.

4 NP-Hardness of Separability

We show k-SEP is NP-hard for a fixed k and signature by a reduction from 3-SAT. We fix k and the signature as this puts verification of a separator in P.[2] Our reduction produces a

[2]This does not by itself imply the problem is in NP, as the separator could be exponentially large.

m_0 +

	X	V_1
X	=	T
V_1	T	=

m_1 −

	X	V_1
X	=	F
V_1	F	=

m_2 +

	X	V_1	V_2	V_3
X	=	A	B	C
V_1	A	=	T	T
V_2	B	T	=	T
V_3	C	T	T	=

m_3 −

	X	V_1	V_2	V_3
X	=	D	E	G
V_1	D	=	F	F
V_2	E	F	=	F
V_3	G	F	F	=

Figure 2. Set of structures $M_{1,1}$ for a formula $(a \lor b \lor c) \land (\neg d \lor \neg e \lor \neg g)$. Each table specifies a structure, giving its polarity (+/−), elements (X, V_1, etc.) and for each pair of elements which binary relation holds. For example, the B in row V_2, column X in m_2 means that $B(X, V_2)$ holds in that structure, and necessarily $\neg A(X, V_2)$, $\neg T(X, V_2)$, etc.

set of structures $M_{k,\ell}$ as a function of the SAT formula, the prefix size k, and the number of universal quantifiers ℓ. We begin by developing intuition using the case where $k = 2$, and then we generalize first to fixed-k-SEP, and then full k-SEP.

4.1 Preprocessing the SAT Problem

We first transform the 3-SAT problem into uniform-3-SAT. If any clauses are not *uniform*, i.e. contain only positive literals or only negative literals, we introduce a fresh variable x to split the clause:

$$(a \lor \neg b \lor c) \mapsto (a \lor c \lor x) \land (\neg x \lor \neg b)$$

In addition, we add fresh variables t and f along with the clauses (t), $(\neg f)$. We will use t and f to represent truth and falsehood. The resulting formula ψ is equisatisfiable with the original problem, and has only uniform clauses with at most three literals.

4.2 Example Construction for ∀∃

We first show the special case of our reduction for the simplest alternating prefix ∀∃. Our structures are defined over a signature which includes binary relations A_i, one for each $a_i \in \text{dom}(\psi)$. Note $t, f \in \text{dom}(\psi)$ so we consider T and F to be part of the A_i but will also refer to these relations directly. All structures will ensure that these relations are *anti-reflexive* ($\neg A_i(x, x)$) and *symmetric* ($A_i(x, y) \Leftrightarrow A_i(y, x)$). Further, for any two distinct elements of the same structure, exactly one of the defined relations or equality will hold between those two elements. Thus the defined relations are *total* (every non-equal pair satisfies some relation) and *mutually-exclusive* (only one relation holds for a pair).

These restrictions mean that for any assignment of x and y, either $x = y$ and no relations hold, or $x \neq y$ and for some i, exactly $A_i(x, y)$ and $A_i(y, x)$ hold. These are the only QF-types (Section 2.3) ever seen by the matrix φ of a separator, and there is one QF-type for $x = y$ and one QF-type for each A_i. Each assignment of x, y determines a QF-type, and the QF-type determines the truth values for the matrix. If we have the same QF-types from different assignments of elements, then the matrix must have the same value. This justifies constructing an assignment from a separator by querying the truth value of φ on the QF-type for A_i, which by abuse of notation we denote $\varphi(A_i)$.

We can now describe the actual structures for a particular input:

$$\psi = (a \lor b \lor c) \land (\neg d \lor \neg e \lor \neg g) \land (t) \land (\neg f)$$

The construction will be to generate a $s + 1$ sized structure with elements $\{X, V_1, \ldots, V_s\}$ for a clause with s literals. The polarity of the structure matches that of the literals in the clause. If we have variables in the clause a_i, a_j, a_k, we will assert $A_i(X, V_1) \land A_i(V_1, X)$, $A_j(X, V_2) \land A_j(V_2, X)$, and $A_k(X, V_3) \land A_k(V_3, X)$. For positive clauses, all other distinct pairs will assert T and negative clauses will assert F. Note that we have used pairs including X to encode the Boolean variables, and those that do not include X get one of T or F (T and F represent Boolean variables in the structures for (t) and $(\neg f)$). We can represent these structures in tabular form, where the rows and columns indicate which structure element corresponds to x and y, while the cell gives the relation (or equality) that holds for that pair. As previously noted, these labels indicate the QF-type for that assignment, and if we have a matrix φ in mind then we can say the *cell itself* is true or false by applying φ. We show the structures for ψ in Figure 2.

With these tables in mind, we can now analyze a separator $\forall x.\exists y.\ \varphi$. In terms of the table, a ∀∃ formula satisfies the structure when every row has a true cell.[3] Similarly, for a negative structure, the separator not satisfying the structure means some row is entirely false. From m_1, we conclude both $\neg\varphi(=)$ and $\neg\varphi(F)$. In m_0, we need one of $(=)$ or T to be true, but since it cannot be $(=)$, we know $\varphi(T)$.

Now we inspect m_2, the structure for $(a \lor b \lor c)$. The bottom three rows all have a T, and so are trivially satisfied. The first row needs a true cell, and therefore one of $\varphi(A)$, $\varphi(B)$, or $\varphi(C)$ holds which is exactly the same constraint as the original clause. In m_3, the structure for $(\neg d \lor \neg e \lor \neg g)$, one of the rows must be entirely false. Regardless of which row is picked, one of $\varphi(D)$, $\varphi(E)$, or $\varphi(G)$ must be false and the negative clause constraint is enforced.

We now see that if a separator exists, then its matrix must produce a pattern of truth values on the QF-types which gives rise to an assignment satisfying each clause. In the other direction, we can easily construct a separator from a satisfying assignment by letting $\varphi = T(x, y) \lor A_i(x, y) \lor$

[3]Due to symmetry, the tables in Figure 2 are symmetric and so we can reason about either rows or columns.

$A_j(x, y) \ldots$ where A_i is included if a_i is true in the assignment. This shows that ∀∃ separation is NP-hard.

4.3 Example Construction is the Same for ∃∀

A somewhat surprising fact is the same construction of structures in Figure 2 is also separable by the ∃∀ prefix iff ψ is satisfiable. Positive structures must have an entirely true row and negative structures must have a false value in every row. When we look at m_0, m_1, this means that (=) is *true* while as before T is *true* and F is *false*. We can see this change to (=) interacts with the change in prefix so that m_2, m_3 still correspond to their clauses. For example, in m_3, the last three rows are trivially satisfied due to the F's, but the first row must now select one of D, E, or G to be false.

These two prefixes give us a taste of a property of our general construction: the set of structures does not depend on the exact prefix. For $k = 2$, there are only two prefixes with alternation and one set of structures. When we generalize, the structures will depend on the parameter ℓ, the number of universal quantifiers.

4.4 General Construction

We define structures $\mathcal{M}_{k,\ell}(\psi)$ over one k-ary relation symbol A_i, for each Boolean variable $a_i \in \text{dom}(\psi)$. In all structures, the relations A_i are (1) symmetric, (2) anti-reflexive, (3) total, and (4) mutually-exclusive.

Each structure $M \in \mathcal{M}_{k,\ell}(\psi)$ will be constructed by a function $C_{k,\ell}(c)$ of a clause $c \in \psi$. The polarity of M is that of the literals in c. Let s be the number of literals in c, so $1 \le s \le 3$. The domain $\text{dom}(M) = X \cup Y \cup V$ will have $k - 1 + s$ elements, labeled X_i, Y_i, and V_j, with cardinality of each depending on the polarity as follows:

	$\lvert X\rvert$	$\lvert Y\rvert$	$\lvert V\rvert$
+	ℓ	$k-1-\ell$	s
-	$\ell-1$	$k-\ell$	s

There will always be exactly $k - 1$ total X and Y elements, collectively the *auxiliary* elements, and one V_j element for each literal, the *variable* elements.[4] Depending on k and ℓ, it is possible that $|X| = 0$ or $|Y| = 0$. We use the clause to associate both a Boolean variable and thus a relation with each V_j, and we label the V_j with the j of this corresponding relation A_j. We then use the following function to assign a relation A_i (or one of T,F) to sets of exactly k elements (k-sets) S:

$$R(S) = \begin{cases} A_j & \text{if } S = X \cup Y \cup \{V_j\} \\ F & \text{else if } X \subseteq S \\ T & \text{otherwise} \end{cases} \tag{1}$$

[4]Note that the positive structure has the same number of X_i as the universal quantifiers and negative structures have the same number of Y_i as existentials.

These rules say if a k-set contains all X and Y, and thus necessarily exactly one V_i, then the set will be assigned a A_j that matches that V_j. If they are not one of these variable k-sets, they will be F if they contain all X_i and T otherwise. This completes our specification of $\mathcal{M}_{k,\ell}(\psi)$.

4.5 Extracting a Satisfying Assignment from a Separator

We assume $M_{k,\ell}$ is separable by some formula Φ, which in prenex normal form is written with the given prefix of k quantifiers and ℓ universals and has matrix φ. We will construct our assignment $\mathcal{A}$ by $\mathcal{A}[a_i] = \varphi(A_i)$, and our goal is to show that this assignment satisfies ψ.

A first observation is that the QF-type for any assignment of k variables to distinct elements in any structure is one of the A_i. A second is that if the winning player plays with the winning strategy, the game always ends in an assignment and QF-type, on which the matrix φ assigns polarity of the structure itself. For example, in positive structures Φ being a separator implies ∃ plays so the matrix is always true. By carefully constructing a strategy for ∀, we will show the game must always end in the k-set for a variable in the clause, showing a variable from the clause must be true in $\mathcal{A}$.

First we need to show that we can force the game to end in a k-set, using the following definition and lemma:

Definition 4.1. A strategy for a prenex formula is *uniqueness-preserving* if, when it plays x_{n+1} in position $(x_1, \ldots, x_n)$, $\text{distinct}(x_1, \ldots, x_n) \Rightarrow \text{distinct}(x_1, \ldots, x_n, x_{n+1})$

Another way to state this is that a strategy is uniqueness-preserving if it is never the first to play a repeated element. Winning strategies for separators of our structures must be uniqueness-preserving:[5]

Lemma 4.2. *If $M^+, M^- \in M_{k,\ell}$, Φ is a prenex formula with k quantifiers, $M^+ \models \Phi$, and $M^- \models \neg\Phi$, then all winning strategies for ∃ in M^+ and ∀ in M^- are uniqueness-preserving.*

Proof. Consider any pair of positive and negative structures M^+ and M^-, and logical games played on both simultaneously, with ∃ in M^+ and ∀ in M^- playing with winning strategies. On each move of this combined game, the winning player plays in one subgame and the losing player then plays in the other according to the prefix. Assume the losing player always *mirrors* the equality of the winning player: either both play new, distinct elements, or both play elements in their respective structure that repeat the same prior variable. If either winning strategy is not uniqueness-preserving, then the games end with the same equalities between variables, and thus the same QF type. But a winning strategy means that this common QF-type must be false in M^- and true in M^+, which is a contradiction. □

[5]Note that it is always possible for a strategy to play distinct elements because our games have at most k moves and there are always at least k elements.

We present a lemma which characterizes T and F, followed by lemmas that show the clause structures constrain φ like ψ constrains $\mathcal{A}$:

Lemma 4.3. *If $\Phi = Q_1x_1. \cdots Q_kx_k.\varphi$ is a k-prenex formula separating $M_{k,\ell}$, then $\varphi(T)$ and $\neg\varphi(F)$.*

Proof. Consider the structures $C(t)$ and $C(\neg f)$. Each has exactly k elements, so if $\forall$ (or respectively $\exists$) picks any available distinct element on its turn, then by Lemma 4.2, the game will end on the only k-set in each structure. But this set corresponds to T in the positive structure and to F in the negative structure, so the lemma holds. □

Lemma 4.4. *If Φ separates $M_{k,\ell}$ and positive clause $c \in \psi$, then $\varphi(A_a)$ holds, for some $a \in c$.*

Proof. Consider the positive structure $C(c)$. $\exists$ must have a winning strategy, and consider the $\forall$ strategy: "play any X_i if not played, otherwise any remaining unplayed element." Both of these strategies are uniqueness-preserving, so by Lemma 4.2 the game will end in a k-set. $\forall$ has ℓ moves, and thus all X will be played. By equation 1, the resulting k-set is either a variable set for some A_a or is assigned F. By Lemma 4.3, it will not end in F because this would make the matrix false, and so $\varphi(A_a)$ holds. □

Lemma 4.5. *If Φ separates $M_{k,\ell}$ and negative clause $c \in \psi$, then $\neg\varphi(A_a)$ holds, for some $\neg a \in c$.*

Proof. Consider the negative structure $C(c)$ and the strategy for $\exists$: "play any Y_i if not played, otherwise any unplayed element." $\exists$ has $k - l$ moves, and so all Y will be played. If not all X are played by some player, then the game ends on T, which is a contradiction. Thus all of X and Y are played, and so the games ends on some variable set, say that of A_a, and thus $\neg\varphi(A_a)$. □

We can now state and prove our desired result:

Theorem 4.6. *If Φ separates $M_{k,\ell}$, then there exists assignment $\mathcal{A}$ which satisfies ψ.*

Proof. Let $\mathcal{A}[a_i] = \varphi(A_i)$ for all Boolean variables a_i. By Lemma 4.3, $\mathcal{A}[t] = \top$ and $\mathcal{A}[f] = \bot$, so clauses (t) and $(\neg f)$ are satisfied. By Lemma 4.4, all other positive clauses have a true variable and by Lemma 4.5, remaining negative clauses have a false variable. Thus $\mathcal{A}$ satisfies ψ. □

4.6 Constructing a Separator from a Satisfying Assignment

We assume that ψ has satisfying assignment $\mathcal{A}$, and we are given a prefix $P = Q_0x_0. \dots Q_{k-1}x_{k-1}$ with k quantifiers and ℓ universals. We then construct a formula $\Phi = P.\ \varphi$ that separates $M_{k,\ell}(\psi)$.

Because the prefix P is given, we need only specify the matrix φ, which will be a disjunction of cases. We add a disjunct $A(x_0, \dots, x_{k-1})$ for each Boolean variable a where $\mathcal{A}[a]$ is true. Note due to the clauses t and $\neg f$ in ψ, $T(\dots)$ will always be a disjunct and $F(\dots)$ will never be. To cover the case when two bound variables are the same element, we add a subset of the formula:

$$E_i = \text{distinct}(x_0, \dots, x_{i-1}) \land (x_0 = x_i \lor x_1 = x_i \lor \cdots \lor x_{i-1} = x_i)$$

E_i expresses the fact that the variables bound before x_i are all distinct, but x_i is equal to one of them. Equivalently, E_i says that x_i is the *first* repeated bound variable. We let $E_0 \equiv \bot$ and note the E_i are mutually exclusive with each other and the relations $A_i(\dots)$. We add E_i as a disjunct if $Q_i = \forall$, and omit E_i otherwise. This completes the specification of φ, which consists of a disjunction of some of the E_i and the relations of variables $\mathcal{A}$ assigns true.

We now show that Φ is true for positive structures in $M_{k,\ell}$ and false for negative structures. We do so by giving strategies for $\exists$ in positive structures and $\forall$ in negative structures such that the game ends with a matrix of the correct polarity.

Lemma 4.7. *$C(c) \models \Phi$ for a positive structure $C(c) \in M_{k,\ell}$*

Proof. We know $\mathcal{A}$ satisfies ψ, so one of the variables $a \in c$ satisfies $\mathcal{A}[a]$ and thus $A(\dots) \in \varphi$. To show Φ is true we make the game end in A or T. Let the element of $C(c)$ which represents a be V_a. Our $\exists$ strategy will be: "play the first unplayed element from V_a, any Y, or any X, in that order." Note that our strategy is uniqueness-preserving. If $\forall$ plays a repeated element, then the corresponding E_i in φ is true regardless of the rest of the played elements, and the matrix is true.

If all played elements are distinct, then note as $\exists$ with $k - \ell$ moves we have ensured that V_a and all Y are played. If all X are played, then we end on A. If not, we end on T. Both relations are in φ, and so the matrix is true. Thus, we can always force the game to end with a true matrix, and so our strategy is winning for $\exists$ and the formula is true. □

Lemma 4.8. *$C(c) \not\models \Phi$ for a negative structure $C(c) \in M_{k,\ell}$*

Proof. $\mathcal{A}$ assigns one of the variables $\neg a \in c$ false, so Φ is false if as $\forall$ we can force the game to end in A or F, because neither will be in φ. Now if $\exists$ plays a repeated element then one of the E_i will be true, but now this time that E_i is omitted from φ. But because all E_i and A are mutually exclusive, no disjunct will be true and the matrix is false regardless of future moves. Now we give a strategy for $\forall$ as: "play the first unplayed of V_a, any X, or any Y, in that order." Now our strategy ensures, due to our ℓ moves, that all of X and V_a will be played. Thus the game ends either in A or F, both of which are absent from φ, and so the matrix will always be false and thus the formula is not true. □

Theorem 4.9. *If ψ has satisfying assignment $\mathcal{A}$, Φ as defined above separates $M_{k,\ell}(\psi)$*

Proof. Follows from Lemmas 4.7 and 4.8. □

4.7 Extending to k-SEP

We have shown that given a particular prefix, we can reduce SAT to separability. However, this does not immediately imply that k-SEP is also difficult, because now the prefix is not given and the extra flexibility might make the problem easier. We extend our construction slightly by adding structures for the trivially satisfied clauses $(t \vee f)$ and $(\neg t \vee \neg f)$. Then we can prove the following lemmas.

Lemma 4.10. *If Φ is a prenex formula with at most k quantifiers and Φ separates $M_{k,\ell}$, then Φ has exactly k quantifiers.*

Proof. Assume for sake of contradiction that Φ has fewer than k quantifiers. Then it does not have k distinct terms, so all defined predicates are false in the matrix φ. Thus the matrix φ is logically equivalent to one which only uses equality, and such a formula cannot distinguish two structures of opposite polarity but equal cardinality, such as $C(t)$ and $C(\neg f)$. Thus, Φ cannot be a separator and we have a contradiction. □

Lemma 4.11. *If Φ is a k-prenex formula separating $M_{k,\ell}$, then Φ must have ℓ universals.*

Proof. Assume for sake of contradiction Φ has $u > \ell$ universals. Consider the positive structure $C(t \vee f)$ and $\forall$ strategy "pick V_f if available, then any X_i, then any Y_i." $\forall$ has at least $\ell + 1$ moves, and so the element V_f as well as all X will be picked. If all $k - 1$ auxiliary elements are picked, then the variable set corresponding to F will have been picked. If not all $k - 1$ auxiliary elements are picked, then because all X_i were, the k-set will still be that of F. Thus the game will always end in F, which is a contradiction.

Assume for sake of contradiction Φ has $u < \ell$ universals. Consider the negative structure $C(\neg t \vee \neg f)$ and $\exists$ strategy "pick V_t if available, then any Y_i, and finally X_i." Because this is a negative structure, $\exists$ has $k - u \geq k - \ell + 1$ moves and element V_t and all Y are picked. If all the X are picked by some player then the game ends in the variable set for T. Otherwise one of the X is missing, and so the k-set is assigned T. The game always ending in T is a contradiction for a negative structure. □

Together, these lemmas establish that any separator must have the same number of universal and existential quantifiers as suggested by k and ℓ. Note the only assumption about the prefix in Lemmas 4.4 and 4.5 was that there were ℓ universals and $k - \ell$ existentials, which we have now established. This means that Theorem 4.6 is also true for k-SEP. Note that the separator Φ can have any prefix with the right number of existentials. For the other direction, we do not need to modify any reasoning because the construction of a formula already made no assumptions about the prefix, and we can use that reasoning to construct separators with any permutation of quantifiers in the prefix.

```
def separate(structures):
    prefix ← ""
    while size of prefix ≤ $k$ do
        if check_prefix(prefix, structures) then
            return build_matrix(prefix, structures)
        prefix ← next_prefix(prefix)
    return ⊥
def check_prefix(prefix, structures):
    $F$ ←
      {if $m$ is positive then sat_formula(prefix, [], $m$)
                          else ¬sat_formula(prefix, [], $m$)
       for $m \in$ structures}
    return $\bigwedge F$ is SAT?
// For prefix $p$, assignment $\sigma$, structure $m$
def sat_formula($p$, $\sigma$, $m$):
    if $p$ is empty then
        return SAT variable of QF-type of $\sigma$ in $m$
    $(Q\, v : S.\ rest) \leftarrow p$
    $f$ ← {sat_formula($rest$, $\sigma \cup [e/v]$, $m$)
           for $e \in m$ of sort $S$}
    return if $Q = \forall$ then $(\bigwedge f)$ else $(\bigvee f)$
```

Figure 3. Pseudocode for the separation algorithm including supporting functions.

4.8 Complexity Results

With Theorems 4.6 and 4.9, we have now shown there is a construction $M_{k,\ell}(\psi)$ that shows fixed-k-SEP as well as k-SEP are NP-hard. We will shortly give an algorithm for solving k-SEP with an oracle for SAT, which will show that k-SEP is in NP, and thus NP-complete. In our separation algorithm, we will actually be solving many fixed-k-SEP instances to solve k-SEP.

5 Separation Algorithm

We now give an algorithm for separation that reduces to SAT, shown in Figure 3. The algorithm enumerates prefixes by size up to size k, and checks whether the given structures can be separated by each one, exiting early if one is found. To check a prefix, the algorithm relies on the fact that the truth of $M \models p$ for a candidate k-prefix separator p is a function of the truth values of the matrix of p for the QF-types of all assignments of the quantified variables to elements of M. To determine if a separator exists, it is sufficient to ask whether there is a propositional assignment to *QF-type variables a_i* such that the separator has the correct polarity on each structure. A key observation here is that the same QF-type may appear in different structures, and that the matrix of any separator must have a consistent truth value for these QF-types, as encoded in the variables a_i.

The sat_formula function recursively constructs a propositional formula which is true precisely when $M \models p$, given the variables a_i. For each quantifier we compute all possible assignments of its variable, and recursively compute the formula with that assignment and the remaining suffix of the formula prefix. When we reach the matrix, we compute and return the QF-type variable a_i for that assignment. We compute a b-bounded QF-type for M, σ by enumerating all finitely many well-sorted atoms a, and record whether $M \cup \sigma \models a$. The number of such atoms is a constant given the signature, b, and k. We then combine the recursive results with $\wedge$ if the quantifier is $\forall$ and $\vee$ if the quantifier is $\exists$. For example, if we have a structure with one sort and two elements x, y, we are checking the prefix $\forall\forall\exists$, and q computes the QF-type index, then we get:

$$\Big(\big(a_{q(x,x,x)} \vee a_{q(x,x,y)}\big) \wedge \big(a_{q(x,y,x)} \vee a_{q(x,y,y)}\big) \Big)$$
$$\wedge \Big(\big(a_{q(y,x,x)} \vee a_{q(y,x,y)}\big) \wedge \big(a_{q(y,y,x)} \vee a_{q(y,y,y)}\big) \Big)$$

With such formulas, we construct a SAT query as the conjunction of all the positive formulas and negation of the negative formulas. This reduction shows that k-SEP is in NP. The remainder of this section discusses how to produce a concrete separator from this reduction, as well as practical heuristics and optimizations.

5.1 Building a Matrix

Given that a particular prefix admits a separator, we show how to build a matrix in conjunctive normal form. Our matrix will consist of some number of clauses of literals, and we can introduce variables $y_{j,k}$ to mean literal j appears in clause k. We require that each atom appears at most once per clause, i.e. $\neg y_{j,k} \vee \neg y_{j',k}$ for j, j' of the same atom. If we let $T_{i,j}$ mean that literal j is true in QF-type i, then we can construct a formula which constrains the QF-type variable to have the same truth value as the matrix in any assignment with that QF-type:

$$\begin{aligned} a_i \leftrightarrow\ & (T_{i,0} \wedge y_{0,0} \vee T_{i,1} \wedge y_{1,0} \vee \ldots) \\ & \wedge (T_{i,0} \wedge y_{0,1} \vee T_{i,1} \wedge y_{1,1} \vee \ldots) \\ & \wedge \ \ldots \end{aligned}$$

Because the T's are constants, this formula will simplify by effectively dropping some of the y terms in each clause, which will be the same between clauses of a_i but different in another $a_{i'}$. If we have an assignment to a_i, we can use this to give us an assignment to $y_{j,k}$, but this can result in overly complex matrices. Instead, we can then add these constraints along with F, and ask for an assignment with a minimal number of $y_{j,k}$. We can also attempt this process with one clause, only increasing it if we get UNSAT, which has the effect of minimizing the number of clauses.

5.2 Limiting Matrix Complexity

We extend this algorithm by introducing a bound w on the number of clauses in the matrix. Even if the SAT query in check_prefix indicates a matrix exists, we eliminate a prefix if the resulting matrix would have too many clauses. As a heuristic, we explore parameters k, w diagonally by increasing their sum $k+w$ whenever we can't separate. This strategy introduces a tradeoff between the number of quantifiers and the number of clauses, rather than considering formulas only by number of quantifiers.

5.3 Lazy Exploration Optimization

One problem with the algorithm as described is that it always computes every assignment to quantified variables, even if not all expansions are necessary. For example, suppose we know that our matrix M satisfies $p(x) \Rightarrow M$. Then if our prefix is $\forall x \forall y \forall z$, in a positive structure once we assign x to an element e that satisfies $p(e)$, we know the formula will be true without considering the assignments of y and z. We modify the algorithm to take advantage of this fact by lazily expanding the set of assignments: every time we get a new proposed matrix that does not actually separate due to unconstrained assignments, we add the constraints which show why that matrix does not work. The SAT query starts with no restrictions and is updated incrementally until either UNSAT is produced, or a correct matrix is found. This optimization is particularly effective if the separation problem can be solved by exploring a small fraction of all assignments.

6 Invariant Inference with Separators

We show how separators can be applied to invariant inference by adapting the IC3/PDR algorithm [4, 9] to use separators and infer general quantified formulas. We first introduce the invariant inference problem, describe a simple ICE learning [14] algorithm and the more advanced IC3/PDR based algorithm, and then discuss the results on a selection of distributed protocols in Section 7.2.

6.1 Invariant Inference

For our purposes a transition system consists of states, a formula *init* describing initial states, a formula *bad* describing bad states, and a transition relation TR between pre- and post-states. We say that if B holds in every post-state of every transition where A holds in the pre-state, then $A \Rightarrow \mathrm{wp}(B)$. Further, the system as a whole is *safe* if the set of bad states is not reachable from the initial states. The invariant inference problem is the problem of establishing safety by inferring a formula I for a given transition system such that:

$$init \Rightarrow I \tag{2}$$
$$I \Rightarrow \mathrm{wp}(I) \tag{3}$$
$$I \Rightarrow \neg bad \tag{4}$$

We say that I is inductive if it satisfies equations (2),(3). The invariant inference problem is then to find an inductive formula that implies safety. For simplicity, we assume the system is safe.

6.2 ICE Learning with Separators

The simplest invariant inference algorithm using separators is ICE learning. The algorithm works by refining a candidate invariant I by incrementally adding constraints derived from counterexamples to equations (2),(3),(4). Positive counterexamples arise from a first-order SAT query of the negation of (2), and negative examples likewise come from (4). Equation (3) is interesting because as observed in [14, 27], counterexamples are *implication constraints*: I only must be true on the post-state if it is true for the pre-state. We note that it is trivial to extend the separation algorithm of Section 5 to support such implication constraints by directly encoding the implication between the formulas for each pair of structures and allowing the SAT solver to handle the disjunction. After a new constraint is added, the separation procedure produces a new I satisfying all known constraints. When no counterexamples exist, the candidate I satisfies equations (2),(3),(4) and the algorithm succeeds.

This algorithm requires learning the invariant monolithically — it is not possible to learn conjuncts of a larger invariant piece by piece. Nevertheless, this algorithm is able to correctly infer the invariant for a few of the the smallest examples from our evaluation (Section 7, Table 2), including ring-id and firewall. The firewall example is notable for requiring an invariant with quantifier alternation.

6.3 IC3/PDR Learning with Separators

To infer the invariants for more complex examples, we need an invariant inference procedure that allows the invariant to be learned incrementally. We describe a version of the IC3/PDR algorithm that uses separators to infer more complex invariants.

The IC3/PDR [4, 9] algorithm maintains a series of conjunctions of formulas $F_0 \dots F_n$ known as *frames*. These frames satisfy the following *frame conditions*:

$$F_0 = init \tag{5}$$
$$F_i \Rightarrow F_{i+1} \tag{6}$$
$$F_i \Rightarrow \mathrm{wp}(F_{i+1}) \tag{7}$$
$$F_i \Rightarrow \neg bad \qquad (\text{for } i < n) \tag{8}$$

Together, these conditions ensure that each frame F_i is an over-approximation of the states reachable in at most i steps from the initial states. Additionally, all frames but the last frame F_n exclude all bad states. As the algorithm runs, new formulas may be added to strengthen a frame, and the algorithm ends when any F_i, $i < n$ is inductive.[6]

To add a new formula, IC3/PDR queries whether there is a bad state in F_n. If there is, that state is *blocked* in frame F_n. If there are no bad states in F_n, then a new frame is opened, initialized to *true*.

To block a state s in a frame F_i, first we recursively block all TR-predecessors of s in the prior frame F_{i-1} that are not already excluded from F_{i-1}.[7] Then we *inductively generalize* by searching for a formula p satisfying:

$$s \not\models p \tag{9}$$
$$init \Rightarrow p \tag{10}$$
$$F_{i-1} \wedge p \Rightarrow \mathrm{wp}(p) \tag{11}$$

Such a p will satisfy initial states, not satisfy s, and be inductive relative to F_{i-1}. Adding p to the current frame and all previous ones (F_j, $1 \leq j \leq i$) will preserve the frame conditions. To find p we can use a very similar procedure to the ICE learning from Section 6.2: we start with *true*, and add constraints that arise from counterexamples to these conditions until we converge to an acceptable p.[8] Because our separation algorithm gives the smallest formula in some sense, the resulting p will be the smallest formula that satisfies the constraints. After a formula is added to some frame, we can apply the standard *pushing* optimization, attempting to move formulas to subsequent frames as long as the frame conditions are still be satisfied. Pushing is important because it prevents the need to rediscover properties in each frame.

7 Evaluation

We evaluate our technique in two ways: first by learning formulas directly from positive and negative examples produced by an oracle, and the second uses separation-based IC3/PDR to infer invariants for distributed protocols.

Our implementation of the separation algorithm uses Z3 [7] to discharge SAT queries, and our adaptation of IC3/PDR is based on the mypyvy framework [12]. Our implementation and benchmarks are publicly available in the artifact of this paper.

7.1 Learning Golden Formulas

To evaluate separation independently of any particular invariant inference procedure, we use a process that *learns* some golden formula G from labeled structures. We start with an empty set of structures, and ask for a separator p.

[6]The algorithm may also end by discovering a concrete sequence of transitions that shows that the system is not safe.

[7]Because all blocked states are either bad states or are known to reach a bad state, if the algorithm tries to block a state in the initial frame F_0 then it can conclude the system is unsafe.

[8]As an optimization, we can add all initial states and transitions with pre-state satisfying F_{i-1} initially rather then requiring them to be inferred again.

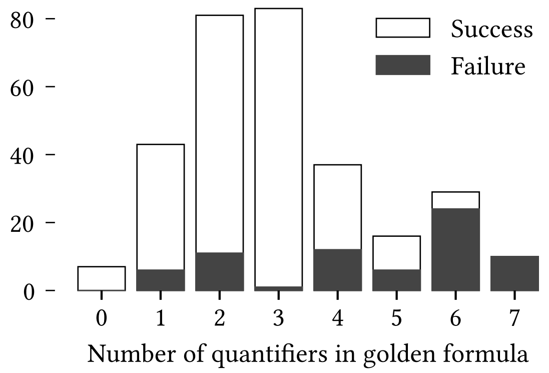

Figure 4. Stacked histogram of learning success (light) and failure (dark) by number of quantifiers in the golden formula.

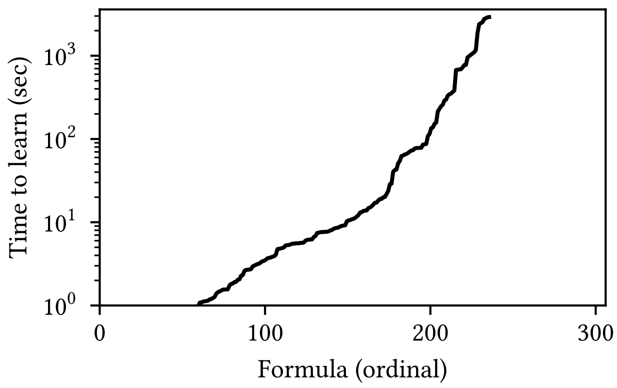

Figure 5. Cactus plot of time to learn formulas. Formulas are ordered by time to learn, and timeouts (>3600 sec) are the blank area on the right.

Then we ask whether $(p \Rightarrow G) \wedge (G \Rightarrow p)$, by querying whether its negation is satisfiable using CVC4 [2], a SMT solver that supports quantifiers and producing models. If the query is UNSAT, then p and G are equivalent, and we have learned G. Otherwise, the solver returns a model M of the query, which is a structure on which p and G differ. We add M to our set of structures labeled according to $M \models G$ and repeat with a new candidate p.

We first describe how we obtained a set of golden formulas, and then discuss the results of running the learning process.

7.1.1 Corpus of Quantified Formulas. We obtained our corpus of golden formulas from the human authored inductive invariants of 27 distributed protocols from previous works [3, 11, 12, 23–25, 29]. Our benchmark examples were chosen to evaluate whether quantified separation is a useful primitive for building invariant inference algorithms, with a focus on examples requiring invariants with quantifier alternations. We present some examples for which our techniques are successful, and some examples that are currently beyond our reach. The protocols are specified in the Ivy and mypyvy systems. Each invariant is manually decomposed into a number of conjuncts, each of which became a golden formula G. The golden formulas are not necessarily in prenex normal form and have anywhere from no quantifiers to 7 quantifiers. We had a total of 306 formulas, and a histogram of formula count by the number of quantifiers can be seen in the total heights of bars in Figure 4.

7.1.2 Results. We ran the learning process with function symbol depth bound $b = 1$ and overall timeout of 1 CPU-core hour for each formula, including the time to explore prefixes, construct minimal matrices, solve for equivalence, and construct counterexamples. Because the constraints only grow, each prefix is eliminated at most once, but multiple formulas of the same prefix are often generated with various matrices before the correct formula is found. The overall success rate of this process is 77.1%, and success rate by number of quantifiers in the golden formula can be seen in Figure 4.

Our results show that separation is successful for most of the formulas up to 5 quantifiers, and less successful for larger sizes. It should be expected that larger prefixes are more difficult to learn. In addition to the exponential scaling caused by quantifier depth, each new quantified variable creates more possible literals that can appear in the matrix. We can observe that 3 quantifiers have a lower failure rate than 1 or 2 quantifiers, which we attribute to other factors affecting the difficulty of learning (e.g. complexity of Boolean matrix or signature) and their distribution in our corpus.

We give a *cactus plot* of the time to learn formulas in Figure 5. In this chart, examples are sorted by their time to learn along the x-axis, while the logarithmic y-axis shows the time to learn that formula. The shape of the chart depends on both the distribution of difficulty in the problems and the performance of the algorithm, so only general trends can be observed. The steep increase around formula 175 suggests additional work is required to handle the most difficult separability problems. As shown by the gap on the right of this chart, our algorithm did not learn about 23% of the corpus, which we analyze next.

7.1.3 Failure Analysis. Closer investigation of the formulas which did not succeed reveals two primary reasons for failure: exponentially many prefixes and the matrix being difficult to infer. Additionally, we observed a single failure because the SMT solver was unable to provide a counterexample. The rarity of this problem, despite generating formulas in first-order logic, is encouraging but it needs to be addressed in any robust system that uses separators.

Some formulas had too many prefixes to explore. This is particularly problematic for some of the 6 and 7 quantifier formulas with several sorts, where a vast majority of the

Table 2. Invariant inference results on examples. Size is the number of conjuncts in the human written invariant. Our solve rate is the percent of 10 runs which were successful within one hour, and the time is the mean of successful runs. PDR$^\forall$ was successful in every run.

Example	∃?	Size	**Our** solve rate	**Our** sec	PDR$^\forall$ sec
ring-id	-	4	100%	38	162
toy-consensus-forall	-	4	100%	10	5
consensus-wo-decide	-	5	100%	63	88
sharded-kv	-	5	100%	20	6
learning-switch	-	6	0%	-	126
consensus-forall	-	7	90%	2341	333
lockserv	-	9	100%	7	13
ticket	-	14	100%	196	224
firewall	Y	2	100%	5	
sharded-kv-no-lost-keys	Y	2	100%	4	
client-server-ae	Y	2	100%	720	
toy-consensus-epr	Y	4	100%	51	
client-server-db-ae	Y	5	0%	-	
ring-id-not-dead	Y	6	70%	612	
consensus-epr	Y	7	90%	890	
hybrid-reliable-broadcast	Y	8	90%	2002	

time was spent eliminating possible prefixes. This difficulty is inherent in the problem, but with more careful engineering or by exploiting the embarrassingly parallel search within a given k, it may be possible to extend our solution to larger prefixes without new algorithmic insights.

A more subtle failure occurs for formulas with matrices with more than one clause. Recall our algorithm finds separators with the minimal sum of the depth and number of clauses, exploring them in a diagonal fashion. For some problems, adding even just one extra quantifier or clause can substantially expand the space of formulas. For one of the examples in our corpus, the algorithm fails because it spends most of the time separating with 5 quantifiers, 1 clause, when the correct solution has 4 quantifiers and 2 clauses. This example finishes quickly if we manually skip 5 quantifier formulas. Solving this problem in general may require better heuristics or a more precise understanding of how the complexity of separation depends on the depth, clauses, and logical signature of the problem.

7.2 IC3/PDR with Separators

We evaluated our implementation of IC3/PDR on the 16 distributed protocols from Section 7.1.1 for which all conjuncts could be individually learned, which ranged from simple (ring-id, lockserv, firewall), to challenging (consensus-epr, hybrid-reliable-broadcast [31]). For comparison, we ran an implementation of the PDR$^\forall$ algorithm for the examples with only universal quantified invariants. We ran each algorithm 10 times to account for randomness in the underlying solvers, and summarize the results in Table 1.

7.2.1 Discussion. On the universal examples, we ran the separators restricted to only universal prefixes, which allows us to do a more direct comparison with the existing PDR$^\forall$ algorithm. We see that the performance of the two algorithms is mostly comparable except for consensus-forall and learning-switch. Unsurprisingly, there is a cost to the generality of our approach. In particular, our algorithm finds formulas with the smallest number of quantifiers first, so consensus-forall it will first find:

$$\forall n.\, \mathrm{votes}(n,\, n) \Rightarrow \mathrm{votemsg}(n,\, n)$$

when it should find:

$$\forall n, n'.\, \mathrm{votes}(n',\, n) \Rightarrow \mathrm{votemsg}(n,\, n')$$

This behavior can waste time discovering formulas that will need to be generalized later.

For the learning-switch example, our algorithm fails to find any invariants. This may be due to the fact this protocol has 3-ary and 4-ary relations with a single sort. Even with just two quantified variables, there are 8 different atoms that a 3-ary relation can generate, and then the matrix can be any of the 2^{2^8} different Boolean functions on this many atoms—and we have not even counted the 4-ary relation. While we don't necessarily need to explore every possible matrix, the extra possibilities slow down separation. This problem would not occur if the arguments of the relation have three different sorts: the same number of atoms requires 6 quantifiers to generate two variables for each sort.

On the examples with quantifier alternation, there is no existing algorithm to compare against. In this case we are able to infer invariants for 7 out of 8 examples. The examples which we are able to infer include both simple (firewall, sharded-kv-no-lost-keys) and complex (consensus-epr) examples. The toy-consensus-epr example is a good example of how protocols require alternation: the core of the invariant is a ∀∃∀ formula expressing that every final consensus value has a quorum of nodes which all voted for that value:

$$\forall v.\, \mathrm{decided}(v) \Rightarrow \exists q.\forall n.\, \mathrm{member}(n,\, q) \Rightarrow \mathrm{vote}(n,\, v)$$

For the client-server-db-ae, there are a few separation queries that dominate the runtime, indicating that our heuristic of minimizing the number of quantifiers and clauses may not be a good fit for this problem. The failures of the ring-id-not-dead example are due to the SMT solver being unable to find a counterexample to equations (10),(11) in the time limit. Although checking if a first-order formula separates a finite set of structures is in fixed-parameter P, checking inductiveness is not decidable in general. A possible solution we leave to future work is to generate separators in a decidable logic fragment.

8 Conclusion

We show that first-order quantified separators are a useful building block in enabling advanced reasoning about quantified formulas. We survey various classes of formulas that could be used as separators, and show that k-prenex normal form has desirable generalization properties. We show that unlike other nearby classes, this class is NP-complete for fixed k by reducing it to and from SAT. Our reduction to SAT results in a practical algorithm for constructing separators from positive and negative examples. We apply our algorithm on a corpus of human authored formulas, which shows that the algorithm is capable of discovering non-trivial formulas. We present the first invariant inference procedure capable of finding invariants with quantifier alternations.

Acknowledgments

We thank Yoni Zohar, our shepherd Shachar Itzhaky, and the anonymous referees for insightful comments which improved this paper. This work was supported by National Science Foundation grants CCF-1160904 and CCF-1409813.

References

[1] Aws Albarghouthi and Kenneth L. McMillan. 2013. Beautiful Interpolants. In *Computer Aided Verification*, Natasha Sharygina and Helmut Veith (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 313–329.

[2] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovi'c, Tim King, Andrew Reynolds, and Cesare Tinelli. 2011. CVC4. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV '11) (Lecture Notes in Computer Science)*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.), Vol. 6806. Springer, 171–177. Snowbird, Utah.

[3] Idan Berkovits, Marijana Lazic, Giuliano Losa, Oded Padon, and Sharon Shoham. 2019. Verification of Threshold-Based Distributed Algorithms by Decomposition to Decidable Logics. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II*. 245–266. https://doi.org/10.1007/978-3-030-25543-5_15

[4] Aaron R. Bradley. 2011. SAT-Based Model Checking without Unrolling. In *Verification, Model Checking, and Abstract Interpretation*, Ranjit Jhala and David Schmidt (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 70–87.

[5] Nader H. Bshouty. 1995. Exact Learning Boolean Functions via the Monotone Theory. *Inf. Comput.* 123, 1 (Nov. 1995), 146–153. https://doi.org/10.1006/inco.1995.1164

[6] Stephen A. Cook. 1971. The Complexity of Theorem-proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing* (Shaker Heights, Ohio, USA) *(STOC '71)*. ACM, New York, NY, USA, 151–158. https://doi.org/10.1145/800157.805047

[7] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (Budapest, Hungary) *(TACAS'08/ETAPS'08)*. Springer-Verlag, Berlin, Heidelberg, 337–340. http://dl.acm.org/citation.cfm?id=1792734.1792766

[8] Samuel Drews and Aws Albarghouthi. 2016. Effectively Propositional Interpolants. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II (Lecture Notes in Computer Science)*, Swarat Chaudhuri and Azadeh Farzan (Eds.), Vol. 9780. Springer, 210–229. https://doi.org/10.1007/978-3-319-41540-6_12

[9] Niklas Eén, Alan Mishchenko, and Robert K. Brayton. 2011. Efficient implementation of property directed reachability. In *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*. 125–134. http://dl.acm.org/citation.cfm?id=2157675

[10] Yotam M. Y. Feldman, Neil Immerman, Mooly Sagiv, and Sharon Shoham. 2020. Complexity and information in invariant inference. *PACMPL* 4, POPL (2020), 5:1–5:29. https://doi.org/10.1145/3371073

[11] Yotam M. Y. Feldman, Oded Padon, Neil Immerman, Mooly Sagiv, and Sharon Shoham. 2017. Bounded Quantifier Instantiation for Checking Inductive Invariants. In *Tools and Algorithms for the Construction and Analysis of Systems*, Axel Legay and Tiziana Margaria (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 76–95.

[12] Yotam M. Y. Feldman, James R. Wilcox, Sharon Shoham, and Mooly Sagiv. 2019. Inferring Inductive Invariants from Phase Structures. In *Computer Aided Verification*, Isil Dillig and Serdar Tasiran (Eds.). Springer International Publishing, Cham, 405–425.

[13] GW Ford and GE Uhlenbeck. 1957. Combinatorial problems in the theory of graphs. In *Proc Natl Acad Sci USA*. 163–167. https://doi.org/10.1073/pnas.43.1.163

[14] Pranav Garg, Christof Löding, P. Madhusudan, and Daniel Neider. 2014. ICE: A Robust Framework for Learning Invariants. In *Computer Aided Verification*, Armin Biere and Roderick Bloem (Eds.). Springer International Publishing, Cham, 69–87.

[15] Jaakko Hintikka. 1982. Game-theoretical semantics: insights and prospects. *Notre Dame J. Formal Logic* 23, 2 (04 1982), 219–241. https://doi.org/10.1305/ndjfl/1093883627

[16] Neil Immerman. 1999. *Descriptive Complexity*. Springer.

[17] Neil Immerman and Eric Lander. 1990. *Describing Graphs: A First-Order Approach to Graph Canonization*. Springer-Verlag, 59–81. https://doi.org/10.1007/978-1-4612-4478-3_5

[18] Daniel Jackson. 2012. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press.

[19] Aleksandr Karbyshev, Nikolaj Bjørner, Shachar Itzhaky, Noam Rinetzky, and Sharon Shoham. 2017. Property-Directed Inference of Universal Invariants or Proving Their Absence. *J. ACM* 64, 1, Article 7 (March 2017), 33 pages. https://doi.org/10.1145/3022187

[20] K. Rustan M. Leino. 2013. Developing Verified Programs with Dafny. In *Proceedings of the 2013 International Conference on Software Engineering* (San Francisco, CA, USA) *(ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 1488–1490. http://dl.acm.org/citation.cfm?id=2486788.2487050

[21] Kenneth L. McMillan. 2003. Interpolation and SAT-Based Model Checking. In *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings (Lecture Notes in Computer Science)*, Warren A. Hunt Jr. and Fabio Somenzi (Eds.), Vol. 2725. Springer, 1–13. https://doi.org/10.1007/978-3-540-45069-6_1

[22] Kenneth L. McMillan. 2006. Lazy Abstraction with Interpolants. In *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings (Lecture Notes in Computer Science)*, Thomas Ball and Robert B. Jones (Eds.), Vol. 4144. Springer, 123–136. https://doi.org/10.1007/11817963_14

[23] Oded Padon, Jochen Hoenicke, Giuliano Losa, Andreas Podelski, Mooly Sagiv, and Sharon Shoham. 2017. Reducing Liveness to Safety in First-Order Logic. *Proc. ACM Program. Lang.* 2, POPL, Article 26 (Dec. 2017), 33 pages. https://doi.org/10.1145/3158114

[24] Oded Padon, Giuliano Losa, Mooly Sagiv, and Sharon Shoham. 2017. Paxos made EPR: decidable reasoning about distributed protocols. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (Oct 2017), 1–31. https://doi.org/10.1145/3140568

[25] Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. 2016. Ivy: Safety Verification by Interactive Generalization. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Santa Barbara, CA, USA) *(PLDI '16)*. ACM, New York, NY, USA, 614–630. https:

//doi.org/10.1145/2908080.2908118

[26] Rahul Sharma and Alex Aiken. 2014. From Invariant Checking to Invariant Inference Using Randomized Search. In *Proceedings of the 16th International Conference on Computer Aided Verification - Volume 8559*. Springer-Verlag, Berlin, Heidelberg, 88–105. https://doi.org/10.1007/978-3-319-08867-9_6

[27] Rahul Sharma, Saurabh Gupta, Bharath Hariharan, Alex Aiken, and Aditya V. Nori. 2013. Verification as Learning Geometric Concepts. In *Static Analysis*, Francesco Logozzo and Manuel Fähndrich (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 388–411.

[28] Rahul Sharma, Aditya V. Nori, and Alex Aiken. 2014. Bias-variance Tradeoffs in Program Analysis. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Diego, California, USA) *(POPL ’14)*. ACM, New York, NY, USA, 127–137. https://doi.org/10.1145/2535838.2535853

[29] Marcelo Taube, Giuliano Losa, Kenneth L. McMillan, Oded Padon, Mooly Sagiv, Sharon Shoham, James R. Wilcox, and Doug Woos. 2018. Modularity for Decidability of Deductive Verification with Applications to Distributed Systems. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Philadelphia, PA, USA) *(PLDI 2018)*. Association for Computing Machinery, New York, NY, USA, 662–677. https://doi.org/10.1145/3192366.3192414

[30] V. Vapnik and A. Chervonenkis. 1971. On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Theory of Probability & Its Applications* 16, 2 (1971), 264–280. https://doi.org/10.1137/1116025 arXiv:https://doi.org/10.1137/1116025

[31] Josef Widder and Ulrich Schmid. 2007. Booting clock synchronization in partially synchronous systems with hybrid process and link failures. *Distributed Computing* 20 (07 2007), 115–140. https://doi.org/10.1007/s00446-007-0026-0