Efficient Implementation of an Abstract Domain of Quantified First-Order Formulas

Eden Frenkel[1](✉), Tej Chajed[2], Oded Padon[3], and Sharon Shoham[1]

[1] Tel Aviv University, Tel Aviv, Israel
edenfrenkel@mail.tau.ac.il
[2] University of Wisconsin-Madison, Madison, WI, USA
[3] VMware Research, Palo Alto, CA, USA

Abstract. This paper lays a practical foundation for using abstract interpretation with an abstract domain that consists of sets of quantified first-order logic formulas. This abstract domain seems infeasible at first sight due to the complexity of the formulas involved and the enormous size of sets of formulas (abstract elements). We introduce an efficient representation of abstract elements, which eliminates redundancies based on a novel syntactic subsumption relation that under-approximates semantic entailment. We develop algorithms and data structures to efficiently compute the join of an abstract element with the abstraction of a concrete state, operating on the representation of abstract elements. To demonstrate feasibility of the domain, we use our data structures and algorithms to implement a symbolic abstraction algorithm that computes the least fixpoint of the best abstract transformer of a transition system, which corresponds to the strongest inductive invariant. We succeed at finding, for example, the least fixpoint for Paxos (which in our representation has 1,438 formulas with $\forall^*\exists^*\forall^*$ quantification) in time comparable to state-of-the-art property-directed approaches.

Keywords: Abstract interpretation · First-order logic · Symbolic abstraction · Invariant inference · Quantifier alternation · Least fixpoint

1 Introduction

Recent years have seen significant progress in automated verification based on first-order logic. In particular, quantified first-order formulas have been used to model many systems, their properties and their inductive invariants [1,6,9–11,13–18,20,22,24,26,28,30,31]. Automatic verification in this domain is challenging because of the combination of the complexity of first-order reasoning performed by solvers and the enormous search space of formulas, especially due to the use of quantifiers. Despite these challenges, there are impressive success stories of automatically inferring quantified inductive invariants for complex distributed and concurrent algorithms [9–11,14,15,17,26,30,31].

A. Gurfinkel and V. Ganesh (Eds.): CAV 2024, LNCS 14682, pp. 86–108, 2024.
https://doi.org/10.1007/978-3-031-65630-9_5

Previous works on invariant inference for first-order logic search for invariants in the form of sets of formulas (interpreted conjunctively) from some language of quantified first-order formulas. Each approach fixes some restricted, typically finite (but extremely large) language $\mathcal{L}$, and searches for a set of $\mathcal{L}$-formulas that form an inductive invariant using sophisticated heuristics and algorithmic techniques, such as property-directed reachability (IC3) [14,15], incremental induction [11,26], generalization from finite instances [9,17], and clever forms of pruning and exploration [30,31]. While prior techniques can successfully handle some challenging examples, the accumulation of specially-tailored techniques makes the results computed by these techniques unpredictable, and makes it hard to extend or improve them.

Abstract interpretation [4,5] suggests a more systematic approach for the development of verification algorithms based on logical languages, where we consider sets of $\mathcal{L}$-formulas as elements in an abstract domain. The abstraction of a set of states S in this domain is given by $\alpha(S) = \{\varphi \in \mathcal{L} \mid \forall s \in S.\ s \models \varphi\}$, i.e., the formulas that are satisfied by all states in the set. Algorithms based on abstract interpretation are better understood and are easier to combine, extend, and improve. However, an abstract domain of quantified first-order formulas seems infeasible: for interesting systems, the abstract elements involved in proofs would contain an astronomical number of formulas.

The main contribution of this work is to develop algorithms and data structures that make an abstract domain based on quantified first-order formulas feasible. Working with this abstract domain introduces two main challenges: (i) efficiently storing and manipulating abstract elements comprising of many formulas, and (ii) overcoming solver limitations when reasoning over them. This work focuses on the first challenge and adopts ideas from prior work [15] to deal with the second. Our techniques lay a practical foundation for using an abstract interpretation approach to develop new analyses in the domain of quantified first-order formulas. We demonstrate feasibility of the abstract domain by applying it to an analysis of several intricate distributed protocols.

Our first key idea is to design a *subsumption relation* for quantified first-order formulas and use it to represent abstract elements (sets of formulas) more compactly, pruning away some formulas that are redundant since they are equivalent to or are entailed by another formula. Subsumption over propositional clauses (disjunctions of literals) is traditionally used for similar pruning purposes (e.g., [19]), but the generalization to first-order formulas, which include disjunction, conjunction, and quantification, is novel.

The second key ingredient of our approach is a way to manipulate abstract elements in our representation. Rather than implementing the standard operations of α (abstraction) and $\sqcup$ (abstract join), we observe that our subsumption-based representation makes it more natural to directly implement an operation that computes the join of an abstract element a with the abstraction of a given concrete state s, i.e., $a \sqcup \alpha(\{s\})$. This operation can be used to compute the abstraction of a set of states, and can also be used to compute the least fixpoint of the best abstract transformer (in the style of symbolic abstraction [27]). The crux of computing $a \sqcup \alpha(\{s\})$ is to *weaken* the formulas in the representation of a to formulas that are subsumed by them and that s satisfies.

Finally, the third key ingredient of our approach is a data structure for storing a set of formulas, with efficient filters for (i) formulas that a given state does not satisfy, and (ii) formulas that subsume a given formula. This data structure is then used to store abstract elements, and the filters make the implementation of $a \sqcup \alpha(\{s\})$ more efficient.

While the paper presents the ingredients of our approach (subsumption, weakening, and the data structure) sequentially, they are interconnected; they all affect each other in subtle ways, and must be designed and understood together. Specifically, there is an intricate tradeoff between the precision of subsumption, which determines the extent of pruning (and therefore the compactness of the representation), and the complexity of abstract domain operations such as weakening (e.g., for computing $a \sqcup \alpha(\{s\})$). The definitions, algorithms, and data structures we present are carefully crafted to balance these considerations. Our subsumption relation, which approximates entailment, is cheap to compute, eliminates enough redundancy to keep the representation of abstract elements compact, and enables an efficient implementation of the weakening operation.

To evaluate our implementation of the abstract domain, we use it to implement a symbolic abstraction [27] procedure that computes the least fixpoint of the best abstract transformer of a transition system (i.e., the strongest inductive invariant for the transition system in the given language). Our evaluation uses benchmarks from the literature, mostly from safety verification of distributed protocols. While our fixpoint computation algorithm is not fully competitive with property-directed invariant inference approaches that exploit various sophisticated heuristics and optimizations, it does demonstrate that fixpoint computation in our abstract domain is feasible, which is quite surprising given the amount of quantified formulas the domain considers. Our approach successfully computes the least fixpoint for transition systems that previously could only be analyzed using property-directed, heuristic techniques (which do not compute the least fixpoint, but an unpredictable heuristic fixpoint). For example, we succeed at finding the strongest inductive invariant of Paxos as modeled in [24] (which in our representation has 1,438 formulas with $\forall^*\exists^*\forall^*$ quantification, representing orders of magnitude more subsumed formulas).

In summary, this paper makes the following contributions:

1. We develop a compact representation of sets of formulas based on a novel syntactic *subsumption relation*. We make a tradeoff here between the extent of pruning and efficiency, accepting some redundant formulas in exchange for practical algorithms. (Sect. 3)
2. We show how to implement a key operation of *weakening* a formula to be satisfied by a given state, and leverage it to compute the join of an abstract element and the abstraction of a state, when abstract elements are represented using our subsumption-based representation. (Sect. 4)
3. We present a data structure that provides an efficient implementation of operations used in the join computation described above. (Sect. 5)

4. We evaluate the approach by applying it to compute the least fixpoint of the best abstract transformer for several distributed and concurrent protocols from the literature, demonstrating the promise of our approach. (Sect. 6)

The rest of this paper is organized as follows: Sect. 2 introduces definitions and notation, Sects. 3 to 6 present the main contributions outlined above, Sect. 7 discusses related work, and Sect. 8 concludes. The proofs of all theorems stated in the paper are given in [8].

2 Background

First-Order Logic. For simplicity of the presentation, we present our approach for single-sorted first-order logic, although in practice we consider many-sorted logic. The generalization of our methods to many-sorted logic is straightforward.

Given a first-order signature Σ that consists of constant, function and relation symbols, the sets of terms and formulas are defined in the usual way: a *term* t is either a variable x, a constant c or a function application $f(t_1, \ldots, t_n)$ on simpler terms; a *formula* is either an equality between terms $t_1 = t_2$, a relation application $r(t_1, \ldots, t_n)$ on terms, or the result of applying Boolean connectives or quantification. We also include $\bot$ as a formula (that is never satisfied).

Terms and formulas are interpreted over first-order structures and assignments to the (free) variables. Given a first-order signature Σ, a structure $\sigma = (\mathcal{U}, \mathcal{I})$ consists of a *universe* $\mathcal{U}$ and an *interpretation* $\mathcal{I}$ to the symbols in Σ. We denote by $\mathbf{structs}[\Sigma]$ the set of structures of Σ whose universe is a finite set.[1] When considering formulas with free variables V, and given some structure $\sigma = (\mathcal{U}, \mathcal{I})$, an *assignment* $\mu : V \to \mathcal{U}$ maps each variable to an element of the structure's universe. We write $(\sigma, \mu) \models \varphi$ to mean that a structure σ with an assignment μ satisfies a formula φ, and $\psi \models \varphi$ to mean that a formula ψ semantically entails φ, i.e., $(\sigma, \mu) \models \psi$ whenever $(\sigma, \mu) \models \varphi$.

Abstract Interpretation. Abstract interpretation [4,5] is a framework for approximating the semantics of systems. It assumes a concrete domain and an abstract domain, each given by a partially ordered set, $(\mathcal{C}, \sqsubseteq_{\mathcal{C}})$ and $(\mathcal{A}, \sqsubseteq_{\mathcal{A}})$, respectively. These are related via a Galois connection consisting of a monotone abstraction function $\alpha : \mathcal{C} \to \mathcal{A}$ and a monotone concretization function $\gamma : \mathcal{A} \to \mathcal{C}$ satisfying $\alpha(c) \sqsubseteq_{\mathcal{A}} a \iff c \sqsubseteq_{\mathcal{C}} \gamma(a)$ for all $a \in \mathcal{A}$ and $c \in \mathcal{C}$.

In this work we consider logical abstract domains parameterized by a finite first-order language $\mathcal{L}$ of closed formulas over signature Σ. In this context, concrete elements are sets of states from $\mathbb{S} = \mathbf{structs}[\Sigma]$,[2] i.e., $\mathcal{C} = \mathcal{P}(\mathbb{S})$, ordered by $\sqsubseteq_{\mathcal{C}}=\subseteq$ (set inclusion). Abstract elements are sets of formulas from $\mathcal{L}$, i.e., $\mathcal{A} = \mathcal{P}(\mathcal{L})$, ordered by $\sqsubseteq_{\mathcal{A}}=\supseteq$, and the Galois connection is given by $\alpha(S) = \{\varphi \in \mathcal{L} \mid \forall s \in \mathbb{S}.\ s \models \varphi\}$ and $\gamma(F) = \{s \in \mathbb{S} \mid \forall \varphi \in F.\ s \models \varphi\}$. That is, abstraction in this domain consists of all $\mathcal{L}$-formulas that hold on a given

[1] We restrict our attention to FOL fragments that have a finite-model property.

[2] Later we consider non-closed formulas and let $\mathbb{S}$ denote structures with assignments.

concrete set, and concretization consists of all states that satisfy a given set of formulas. Note that sets of formulas are interpreted conjunctively in this context.

This logical abstract domain forms a join-semilattice (meaning every two elements have a least upper bound) with a least element. The least element, denoted $\bot_{\mathcal{A}}$ (not to be confused with the formula $\bot$), is $\mathcal{L}$, and join, denoted $\sqcup$, corresponds to set intersection. For example, $F \sqcup \alpha(\{s\}) = F \cap \{\varphi \in \mathcal{L} \mid s \models \varphi\} = \{\varphi \in F \mid s \models \varphi\}$, and can be understood as *weakening* F by eliminating from it all formulas that are not satisfied by s.

3 Subsumption-Based Representation of Sets of Formulas

In this section we develop an efficient representation for elements in the abstract domain $\mathcal{A} = \mathcal{P}(\mathcal{L})$ induced by a finite first-order language $\mathcal{L}$. The abstract elements are sets of formulas, interpreted conjunctively, which may be extremely large (albeit finite). Our idea is to reduce the size and complexity of such sets by avoiding redundancies that result from semantic equivalence and entailment. For example, when representing a set of formulas we would like to avoid storing both φ and ψ when they are semantically equivalent ($\varphi \equiv \psi$). Similarly, if $\varphi \models \psi$ then instead of keeping both φ and ψ we would like to keep only φ.

In practice, it is not possible to remove all such redundancies based on semantic equivalence and entailment, since, as we shall see in Sect. 4, performing operations over the reduced representation of abstract elements involves recovering certain subsumed formulas, and finding these in the case of entailment essentially requires checking all formulas in the language. This is clearly infeasible for complex languages such as the ones used in our benchmarks (see Table 1), and is exacerbated by the fact that merely checking entailment is expensive for formulas with quantifiers. Instead, our key idea is to remove redundancies based on a cheap-to-compute subsumption relation, which approximates semantic entailment, and enables efficient operations over abstract elements such as joining them with an abstraction of a concrete state.

We start the section with an inductive definition of a family of finite first-order languages that underlies all of our developments (Sect. 3.1). We then introduce a syntactic *subsumption* relation for first-order formulas (Sect. 3.2), which we leverage to develop an efficient *canonicalization* of formulas, effectively determining a single representative formula for each subsumption-equivalence class (Sect. 3.3). We then use antichains of canonical formulas, i.e., sets of canonical formulas where no formula is subsumed by another, to represent sets of formulas (Sect. 3.4). Sects. 4 and 5 develop ways to effectively manipulate this representation in order to accommodate important operations for abstract interpretation algorithms, such as weakening an abstraction to include a given concrete state.

3.1 Bounded First-Order Languages

At core of our approach is an inductively-defined family of first-order languages, termed *bounded first-order languages*. These languages are all finite and bound various syntactic measures of formulas (e.g., number of quantifiers, size of the Boolean structure), which, in turn, determine the precision of the abstract domain. The inductive definition of bounded languages facilitates efficient recursive implementations of our developments.

We fix a signature Σ and a variable set V. Definition 1 provides the inductive definition of the family of bounded first-order languages (over Σ and V), where each language $\mathcal{L}$ is also equipped with a bottom element $\bot_{\mathcal{L}}$ (equivalent to false). We use $\mathfrak{S}_X$ to denote the set of permutations over a set of variables X, and use $\varphi\pi$ to denote the formula obtained by substituting free variables in a formula φ according to $\pi \in \mathfrak{S}_X$. A set of formulas F is $\mathfrak{S}_X$-closed if $\varphi\pi \in F$ for every $\varphi \in F$, $\pi \in \mathfrak{S}_X$. All bounded first-order languages will be $\mathfrak{S}_V$-closed; this will be important for canonicalization. We use $\bar{\varphi} = \langle \varphi_1, \ldots, \varphi_n \rangle$ to denote a sequence of formulas, φ_{-i} to denote the formula φ_{n-i+1} in the sequence, $|\bar{\varphi}|$ for the length of $\bar{\varphi}$, and $[\bar{\varphi}]$ for its set of indices $\{1, \ldots, |\bar{\varphi}|\}$. We use $\mathcal{L}^*$ for the set of all (finite) sequences of formulas from $\mathcal{L}$, and ϵ for the empty sequence ($|\epsilon| = 0$).

Definition 1 (Bounded First-Order Languages). *A* bounded first-order language *is one of the following, where $X \subseteq V$ denotes a finite set of variables, and $\mathcal{L}$, $\mathcal{L}_1$ and $\mathcal{L}_2$ denote bounded first-order languages:*

$$\begin{aligned}
\mathcal{L}_A &= A \cup \{\bot\} \textit{ with } \bot_{\mathcal{L}_A} = \bot\textit{, where } A \textit{ is any finite } \mathfrak{S}_V\textit{-closed set of formulas}\\
\vee[\mathcal{L}_1, \mathcal{L}_2] &= \{\varphi_1 \vee \varphi_2 \mid \varphi_1 \in \mathcal{L}_1, \varphi_2 \in \mathcal{L}_2\} \textit{ with } \bot_{\vee[\mathcal{L}_1,\mathcal{L}_2]} = \bot_{\mathcal{L}_1} \vee \bot_{\mathcal{L}_2}\\
\wedge[\mathcal{L}_1, \mathcal{L}_2] &= \{\varphi_1 \wedge \varphi_2 \mid \varphi_1 \in \mathcal{L}_1, \varphi_2 \in \mathcal{L}_2\} \textit{ with } \bot_{\wedge[\mathcal{L}_1,\mathcal{L}_2]} = \bot_{\mathcal{L}_1} \wedge \bot_{\mathcal{L}_2}\\
\vee_k[\mathcal{L}] &= \{\bigvee \bar{\varphi} \mid \bar{\varphi} \in \mathcal{L}^* \textit{ and } |\bar{\varphi}| \leq k\} \textit{ with } \bot_{\vee_k[\mathcal{L}]} = \bigvee \epsilon\textit{, where } k \in \mathbb{N}\\
\wedge_\omega[\mathcal{L}] &= \{\bigwedge \bar{\varphi} \mid \epsilon \neq \bar{\varphi} \in \mathcal{L}^*\} \textit{ with } \bot_{\wedge_\omega[\mathcal{L}]} = \bigwedge \langle \bot_{\mathcal{L}} \rangle\\
\exists_X[\mathcal{L}] &= \{\exists X.\varphi \mid \varphi \in \mathcal{L}\} \textit{ with } \bot_{\exists_X[\mathcal{L}]} = \exists X.\bot_{\mathcal{L}}\\
\forall_X[\mathcal{L}] &= \{\forall X.\varphi \mid \varphi \in \mathcal{L}\} \textit{ with } \bot_{\forall_X[\mathcal{L}]} = \forall X.\bot_{\mathcal{L}}\\
\exists\!\!\!\forall_X[\mathcal{L}] &= \{QX.\varphi \mid \varphi \in \mathcal{L}, Q \in \{\exists, \forall\}\} \textit{ with } \bot_{\exists\!\!\!\forall_X[\mathcal{L}]} = \forall X.\bot_{\mathcal{L}}
\end{aligned}$$

The base case is any finite set of formulas (over Σ and V) that is closed under variable permutations, augmented by $\bot$ (denoting false). Typical examples include the set of all literals over Σ and V with a bounded depth of function applications. We introduce binary language constructors for disjunction and conjunction, each operating on two possibly different languages. We also introduce constructors for homogeneous disjunction of at most k disjuncts, as well as unbounded non-empty conjunction, over any single language. Finally, we introduce constructors for quantification ($\exists$ or $\forall$) over a finite set of variables and a language, as well as a constructor that includes both quantifiers for languages where both options are desired. Note that for the construction of a logical abstract domain, we are interested in languages where all formulas are closed (have no free variables), but the inductive definition includes languages with free variables.

The semantics of formulas in each language is defined w.r.t. states $\mathbb{S}$ that consist of first-order structures and assignments to the free variables, following the standard first-order semantics, extended to conjunctions and disjunctions of finite sequences in the natural way, where $\bigvee \epsilon \equiv \bot$. (We do not allow $\bigwedge \epsilon$, which would have been equivalent to "true", since it is not useful for our developments.)

Observe that for a fixed language $\mathcal{L}$, the formulas $\varphi_1 \vee \varphi_2 \in \vee[\mathcal{L}, \mathcal{L}]$ and $\bigvee\langle\varphi_1, \varphi_2\rangle \in \vee_2[\mathcal{L}]$ are syntactically different but semantically equivalent (and similarly for conjunctions). Nonetheless, we introduce homogeneous disjunction and conjunction since they admit a more precise subsumption relation, yielding a more efficient representation of sets of formulas. Also note that we consider bounded disjunction but unbounded conjunction; Sect. 4.3 explains this choice.

Example 1. $\mathcal{L} = \forall_{\{x,y\}}[\vee_2[\mathcal{L}_A]]$ with $A = \{p(x), \neg p(x), p(y), \neg p(y)\}$ is a bounded first-order language over signature Σ that has one unary predicate p and variables $V = \{x, y\}$. Formulas in this language are universally quantified homogeneous disjunctions of at most two literals. For instance, $\mathcal{L}$ includes $\forall\{x, y\}. \bigvee \epsilon$, which is also $\bot_{\mathcal{L}}$, as well as $\forall\{x, y\}. \bigvee\langle p(x)\rangle$, $\forall\{x, y\}. \bigvee\langle p(x), \neg p(y)\rangle$, etc.

3.2 Syntactic Subsumption

Next, we define a *subsumption* relation for each bounded first-order language. The subsumption relation serves as an easy-to-compute under-approximation for entailment between formulas from the same language. We use $\sqsubseteq_{\mathcal{L}}$ to denote the subsumption relation for language $\mathcal{L}$, or simply $\sqsubseteq$ when $\mathcal{L}$ is clear from context. When $\varphi \sqsubseteq \psi$ we say φ *subsumes* ψ, and then we will also have $\varphi \models \psi$.

Definition 2 (Subsumption). *We define $\sqsubseteq_{\mathcal{L}}$ inductively, following the definition of bounded first-order languages, as follows, where $\circ \in \{\vee, \wedge\}$, $k \in \mathbb{N}$, $Q, Q' \in \{\exists, \forall\}$, X is a finite set of variables, and $\mathcal{L}$, $\mathcal{L}_1$ and $\mathcal{L}_2$ are bounded first-order languages:*

$$\varphi \sqsubseteq_{\mathcal{L}_A} \psi \textit{ iff } \varphi = \bot \textit{ or } \varphi = \psi$$

$$\varphi_1 \circ \varphi_2 \sqsubseteq_{\circ[\mathcal{L}_1,\mathcal{L}_2]} \psi_1 \circ \psi_2 \textit{ iff } \varphi_1 \sqsubseteq_{\mathcal{L}_1} \psi_1 \textit{ and } \varphi_2 \sqsubseteq_{\mathcal{L}_2} \psi_2 \quad \textit{(pointwise extension)}$$

$$\bigvee \bar{\varphi} \sqsubseteq_{\vee_k[\mathcal{L}]} \bigvee \bar{\psi} \textit{ iff } \exists m \colon [\bar{\varphi}] \to [\bar{\psi}].\, \forall i \in [\bar{\varphi}].\, \varphi_i \sqsubseteq \psi_{m(i)} \textit{ and } m \textit{ is injective}$$

$$\bigwedge \bar{\varphi} \sqsubseteq_{\wedge_\omega[\mathcal{L}]} \bigwedge \bar{\psi} \textit{ iff } \exists m \colon [\bar{\psi}] \to [\bar{\varphi}].\, \forall i \in [\bar{\psi}].\, \varphi_{m(i)} \sqsubseteq \psi_i$$

$$(QX.\varphi) \sqsubseteq_{Q_X[\mathcal{L}]} (QX.\psi) \textit{ iff } \exists \pi \in \mathfrak{S}_X.\, \varphi \sqsubseteq_{\mathcal{L}} \psi\pi$$

$$(QX.\varphi) \sqsubseteq_{\exists\!\!\!\forall_X[\mathcal{L}]} (Q'X.\psi) \textit{ iff } \exists \pi \in \mathfrak{S}_X.\, \varphi \sqsubseteq_{\mathcal{L}} \psi\pi, \textit{ and } Q = \forall \textit{ or } Q' = \exists$$

The subsumption relation of a bounded first-order language $\mathcal{L}$ is composed, hierarchically, from the subsumption relations of the bounded first-order languages that $\mathcal{L}$ is composed from. For example, the languages participating in the composition of $\mathcal{L} = \forall_{\{x,y\}}[\vee_2[\mathcal{L}_A]]$ defined in Example 1 are $\mathcal{L}_A$, $\vee_2[\mathcal{L}_A]$, and $\forall_{\{x,y\}}[\vee_2[\mathcal{L}_A]]$, and each is equipped with its own subsumption relation.

In the base case, formulas in $\mathcal{L}_A$ are only subsumed by themselves or by $\bot$. For example, considering Example 1, $p(x) \not\sqsubseteq_{\mathcal{L}_A} p(y)$. Subsumption is lifted to languages obtained by binary conjunctions and disjunctions in a pointwise manner. For the languages obtained by homogeneous constructors, a mapping over indices determines which element of one sequence subsumes which element of the other. To approximate entailment, the mapping in the disjunctive case maps each element of $\bigvee \bar{\varphi}$ to one in $\bigvee \bar{\psi}$ that it subsumes, and in the conjunctive case maps each element of $\bigwedge \bar{\psi}$ to one in $\bigwedge \bar{\varphi}$ that subsumes it. As a result, subsumption is more precise in the homogeneous case than in the binary one. For example, considering A from Example 1, $p(x) \vee p(y) \not\sqsubseteq_{\vee[\mathcal{L}_A, \mathcal{L}_A]} p(y) \vee p(x)$, even though the formulas are semantically equivalent. On the other hand, $\bigvee\langle p(x), p(y)\rangle \sqsubseteq_{\vee_2[\mathcal{L}_A]} \bigvee\langle p(y), p(x)\rangle$

In the case of quantifiers, subsumption is lifted from the language of the body while considering permutations over the quantified variables. For example, in Example 1, $\forall\{x, y\}. \bigvee\langle p(x)\rangle \sqsubseteq_{\mathcal{L}} \forall\{x, y\}. \bigvee\langle p(y)\rangle$ due to variable permutations, even though $\bigvee\langle p(x)\rangle \not\sqsubseteq_{\vee_2[\mathcal{L}_A]} \bigvee\langle p(y)\rangle$. When both quantifiers are considered, a universal quantifier can subsume an existential one.

The injectivity requirement for $\sqsubseteq_{\vee_k[\mathcal{L}]}$ can be dropped without damaging any of the definitions or theorems in this section, but it enables a simpler definition of the weakening operator in Sect. 4 (as discussed further in Sect. 4.3).

The following theorem establishes the properties of $\sqsubseteq_{\mathcal{L}}$.

Theorem 1 (Properties of $\sqsubseteq_{\mathcal{L}}$). *For any bounded first-order language $\mathcal{L}$, $\sqsubseteq_{\mathcal{L}}$ is a preorder (i.e., reflexive and transitive) such that for any $\varphi, \psi \in \mathcal{L}$, if $\varphi \sqsubseteq \psi$ then $\varphi \models \psi$. Moreover, $\bot_{\mathcal{L}} \sqsubseteq_{\mathcal{L}} \varphi$ for any $\varphi \in \mathcal{L}$.*

As with entailment, where two distinct formulas can entail each other (i.e., be semantically equivalent), there can be distinct formulas $\varphi, \psi \in \mathcal{L}$ with $\varphi \sqsubseteq_{\mathcal{L}} \psi$ and $\psi \sqsubseteq_{\mathcal{L}} \varphi$ (since $\sqsubseteq_{\mathcal{L}}$ is not always a partial order, i.e., not antisymmetric). We call such formulas *subsumption-equivalent*, and denote this by $\varphi \equiv_{\sqsubseteq_{\mathcal{L}}} \psi$. ($\equiv_{\sqsubseteq_{\mathcal{L}}}$ is clearly an equivalence relation.) The existence of subsumption-equivalent formulas is a positive sign, indicating that our subsumption relation manages to capture nontrivial semantic equivalences. This is thanks to the definition of subsumption for homogeneous disjunction and conjunction, as well as for quantification. For example, $\bigvee\langle\varphi, \psi\rangle \equiv_{\sqsubseteq} \bigvee\langle\psi, \varphi\rangle$ (and similarly for conjunction), and if $\varphi \sqsubseteq \psi$ then $\bigwedge\langle\varphi, \psi\rangle \equiv_{\sqsubseteq} \bigwedge\langle\varphi\rangle$. For quantifiers, $QX.\varphi \equiv_{\sqsubseteq} QX.\varphi\pi$ for any $\pi \in \mathfrak{S}_X$ and $Q \in \{\exists, \forall\}$. (In contrast, $\sqsubseteq_{\mathcal{L}_A}$ is always antisymmetric, and the definitions of $\vee[\mathcal{L}_1, \mathcal{L}_2]$ and $\wedge[\mathcal{L}_1, \mathcal{L}_2]$ preserve antisymmetry.)

3.3 Canonicalization

As a first step towards an efficient representation of sets of formulas, we use a canonicalization of formulas w.r.t. $\equiv_{\sqsubseteq}$, which allows us to only store canonical formulas as unique representatives of their (subsumption-) equivalence class. In general, a *canonicalization* w.r.t. an equivalence relation $\equiv$ over a set S is a function $c\colon S \to S$ such that $\forall x \in S. c(x) \equiv x$ (representativeness) and $\forall x, y \in S. x \equiv y \iff c(x) = c(y)$ (decisiveness). We say that x is *canonical* if $c(x) = x$. When the equivalence relation is derived from a preorder (as $\equiv_{\sqsubseteq}$ is derived from $\sqsubseteq$) then the preorder is a partial order over the set of canonical elements. For our case, that means that $\sqsubseteq_{\mathcal{L}}$ is a partial order over the set of canonical $\mathcal{L}$-formulas.

It is useful, both for the algorithms developed in the sequel and for the definition of canonicalization for $Q_X[\mathcal{L}]$ ($Q \in \{\exists, \forall, \exists\!\!\!\forall\}$), to define a total order

$\leq_{\mathcal{L}}$ over canonical $\mathcal{L}$-formulas that extends $\sqsubseteq_{\mathcal{L}}$. We thus define the canonicalization function $c_{\mathcal{L}}$ and the total order $\leq_{\mathcal{L}}$ over canonical $\mathcal{L}$-formulas by mutual induction. For a set of canonical $\mathcal{L}$-formulas F, we use $\min_{\sqsubseteq_{\mathcal{L}}} F$ to denote the set of formulas in F not subsumed by others, i.e., $\min_{\sqsubseteq_{\mathcal{L}}} F = \{\varphi \in F \mid \forall\psi \in F\backslash\{\varphi\}.\, \psi \not\sqsubseteq \varphi\}$, and use $\min_{\leq_{\mathcal{L}}} F$ to denote the minimal element of a non-empty set F w.r.t. the total order $\leq_{\mathcal{L}}$. Finally, we use $\langle\bar{\varphi}\rangle_{\leq}$ for the sequence obtained by sorting $\bar{\varphi}$ according to $\leq$ in ascending order, and similarly $\langle F\rangle_{\leq}$ for the sequence obtained by sorting the elements of a set F.

Definition 3 (Canonicalization). *For every bounded first-order language $\mathcal{L}$, we define the* canonicalization function $c_{\mathcal{L}} \colon \mathcal{L} \to \mathcal{L}$ *and a total order $\leq_{\mathcal{L}}$ over canonical $\mathcal{L}$-formulas by mutual induction (where $\circ \in \{\vee, \wedge\}$ and $Q \in \{\exists, \forall\}$):*

$$
\begin{aligned}
c_{\mathcal{L}_A}(\varphi) &= \varphi\\
c_{\circ[\mathcal{L}_1,\mathcal{L}_2]}(\varphi_1 \circ \varphi_2) &= c_{\mathcal{L}_1}(\varphi_1) \circ c_{\mathcal{L}_2}(\varphi_2) \ \textit{(pointwise)}\\
c_{\vee_k[\mathcal{L}]}(\bigvee \bar{\varphi}) &= \bigvee \langle c_{\mathcal{L}}(\varphi_1), \ldots, c_{\mathcal{L}}(\varphi_{|\bar{\varphi}|})\rangle_{\leq_{\mathcal{L}}}\\
c_{\wedge_\omega[\mathcal{L}]}(\bigwedge \bar{\varphi}) &= \bigwedge \langle \min\nolimits_{\sqsubseteq_{\mathcal{L}}} \{c_{\mathcal{L}}(\varphi_1), \ldots, c_{\mathcal{L}}(\varphi_{|\bar{\varphi}|})\}\rangle_{\leq_{\mathcal{L}}}\\
c_{Q_X[\mathcal{L}]}(QX.\varphi) &= QX.\min\nolimits_{\leq_{\mathcal{L}}} \left\{c_{\mathcal{L}}(\varphi\pi) \;\middle|\; \pi \in \mathfrak{S}_X\right\}\\
c_{\exists\!\!\forall_X[\mathcal{L}]}(QX.\varphi) &= c_{Q_X[\mathcal{L}]}(QX.\varphi)
\end{aligned}
$$

and

$$
\begin{aligned}
&\leq_{\mathcal{L}_A} \textit{ is an arbitrary total order extending } \sqsubseteq_{\mathcal{L}_A}\\
\varphi_1 \circ \varphi_2 \leq_{\circ[\mathcal{L}_1,\mathcal{L}_2]} \psi_1 \circ \psi_2 &\iff \varphi_1 <_{\mathcal{L}_1} \psi_1, \textit{ or } \varphi_1 = \psi_1 \textit{ and } \varphi_2 \leq_{\mathcal{L}_2} \psi_2\\
\bigvee \bar{\varphi} \leq_{\vee_k[\mathcal{L}]} \bigvee \bar{\psi} &\iff \bar{\varphi} \textit{ is a suffix of } \bar{\psi},\\
&\qquad \textit{or } \exists i \in [\bar{\varphi}] \cap [\bar{\psi}].\ \varphi_{-i} <_{\mathcal{L}} \psi_{-i} \wedge \forall j < i.\ \varphi_{-j} = \psi_{-j}\\
\bigwedge \bar{\varphi} \leq_{\wedge_\omega[\mathcal{L}]} \bigwedge \bar{\psi} &\iff \bar{\psi} \textit{ is a prefix of } \bar{\varphi},\\
&\qquad \textit{or } \exists i \in [\bar{\varphi}] \cap [\bar{\psi}].\ \varphi_i <_{\mathcal{L}} \psi_i \wedge \forall j < i.\ \varphi_j = \psi_j\\
QX.\varphi \leq_{Q_X[\mathcal{L}]} QX.\psi &\iff \varphi \leq_{\mathcal{L}} \psi\\
QX.\varphi \leq_{\exists\!\!\forall_X[\mathcal{L}]} Q'X.\psi &\iff Q = Q' \textit{ and } \varphi \leq_{\mathcal{L}} \psi, \textit{ or } Q = \forall \textit{ and } Q' = \exists
\end{aligned}
$$

where $\varphi <_{\mathcal{L}} \psi$ is shorthand for "$\varphi \leq_{\mathcal{L}} \psi$ and $\varphi \neq \psi$".

Our inductive definition of canonicalization in Definition 3 recognizes the only possible sources of nontrivial subsumption-equivalence in our construction: non-canonicity of subformulas, ordering of sequences, internal subsumption in $\wedge_\omega[\cdot]$-sequences, and permuting of quantified variables. To address these, we canonicalize all subformulas, order their sequences w.r.t $\leq_{\mathcal{L}}$ in $\vee_k[\mathcal{L}]$ and $\wedge_\omega[\mathcal{L}]$, minimize $\wedge_\omega[\mathcal{L}]$-sequences w.r.t $\sqsubseteq_{\mathcal{L}}$, and in $Q_X[\mathcal{L}]$, $Q \in \{\exists, \forall, \exists\!\!\forall\}$, choose the permutation yielding the $\leq_{\mathcal{L}}$-least (canonical) body. For the total order in the cases of Boolean connectives, we use lexicographic-like orderings carefully designed to extend their associated subsumption relations (e.g., homogeneous disjunction uses a right-to-left lexicographic ordering). For quantification, the total order is directly lifted from the total order for canonical bodies.

As an example, consider $\mathcal{L} = \forall_{\{x,y\}}[\vee_2[\mathcal{L}_A]]$ from Example 1. To obtain a canonicalization for $\mathcal{L}$, we provide an arbitrary total order $\leq_{\mathcal{L}_A}$, say $p(x) <_{\mathcal{L}_A} \neg p(x) <_{\mathcal{L}_A} p(y) <_{\mathcal{L}_A} \neg p(y)$ (recall that $\bot \in \mathcal{L}_A$ is least). This uniquely determines the total order and canonicalization of $\mathcal{L}$ and all of its sub-languages. For example, canonicalization of both $\forall\{x,y\}.\bigvee\langle p(x)\rangle$ and $\forall\{x,y\}.\bigvee\langle p(y)\rangle$, which are $\sqsubseteq_{\mathcal{L}}$-equivalent, is $\forall\{x,y\}.\bigvee\langle p(x)\rangle$. This is because $p(x) <_{\mathcal{L}_A} p(y)$, and thus $c_{\vee_2[\mathcal{L}_A]}(\bigvee\langle p(x)\rangle) = \bigvee\langle p(x)\rangle <_{\vee_2[\mathcal{L}_A]} \bigvee\langle p(y)\rangle = c_{\vee_2[\mathcal{L}_A]}(\bigvee\langle p(y)\rangle)$. Note that $\bigvee\langle p(x)\rangle$ and $\bigvee\langle p(y)\rangle$ are both canonical, but adding quantifiers merges the two formulas into the same subsumption-equivalence class, necessarily making the quantified version of one of them non-canonical. Similarly, the $\sqsubseteq_{\vee_2[\mathcal{L}_A]}$-equivalent formulas $\bigvee\langle p(x), p(y)\rangle$ and $\bigvee\langle p(y), p(x)\rangle$ are both canonicalized into $\bigvee\langle p(x), p(y)\rangle$ (by sorting the sequences of literals according to $\leq_{\mathcal{L}_A}$).

The properties of $c_{\mathcal{L}}$ and $\leq_{\mathcal{L}}$ defined above are established by the following theorem, which ensures that Definition 3 is well-defined (e.g., that whenever $\min_{\leq_{\mathcal{L}}}$ is used, $\leq_{\mathcal{L}}$ is a total order).

Theorem 2. *For any bounded language $\mathcal{L}$, $c_{\mathcal{L}}$ is a canonicalization w.r.t. $\equiv_{\sqsubseteq_{\mathcal{L}}}$, that is, it is representative ($c_{\mathcal{L}}(\varphi) \equiv_{\sqsubseteq_{\mathcal{L}}} \varphi$) and decisive ($\varphi \equiv_{\sqsubseteq_{\mathcal{L}}} \psi \iff c_{\mathcal{L}}(\varphi) = c_{\mathcal{L}}(\psi)$); $\sqsubseteq_{\mathcal{L}}$ is a partial order over canonical $\mathcal{L}$-formulas; and $\leq_{\mathcal{L}}$ is a total order over canonical $\mathcal{L}$-formulas that extends $\sqsubseteq_{\mathcal{L}}$.*

Corollary 1. *For any $\varphi, \psi \in \mathcal{L}$, if $\varphi \sqsubseteq_{\mathcal{L}} \psi$ then $c_{\mathcal{L}}(\varphi) \leq_{\mathcal{L}} c_{\mathcal{L}}(\psi)$.*

Henceforth, we use $\boldsymbol{\mathcal{L}}$ to denote the set of canonical $\mathcal{L}$-formulas.

3.4 Representing Sets of Formulas

We utilize the subsumption relation and canonicalization to efficiently represent sets of formulas which are interpreted conjunctively as antichains of canonical formulas, where an *antichain* is a set of formulas incomparable by subsumption.

Definition 4 (Set Representation). *Given a set of formulas $F \subseteq \mathcal{L}$, we define its* representation *as the set $R_F = \min_{\sqsubseteq_{\mathcal{L}}}\{c(\varphi) \mid \varphi \in F\}$.*

The representation combines two forms of redundancy elimination: the use of canonical formulas eliminates redundancies due to subsumption-equivalence, and the use of $\sqsubseteq_{\mathcal{L}}$-minimal elements reduces the size of the set by ignoring subsumed formulas. Observe that the more permissive the subsumption relation is, the smaller the set representations are, because more formulas will belong to the same equivalence class and more formulas will be dropped by $\min_{\sqsubseteq_{\mathcal{L}}}$.

This representation preserves the semantics of a set of formulas (interpreted conjunctively). For sets that are upward-closed w.r.t. subsumption (e.g., $\alpha(S)$ for some set of states S), the representation is lossless as a set can be recovered by taking the upward closure of its representation. For a set $F \subseteq \mathcal{L}$, we use $\uparrow F$ to denote its *upward closure* (w.r.t. $\sqsubseteq_{\mathcal{L}}$), given by $\uparrow F = \{\varphi \in \mathcal{L} \mid \exists \psi \in F.\, \psi \sqsubseteq_{\mathcal{L}} \varphi\}$.

Theorem 3 (Antichain Representation). *For $F \subseteq \mathcal{L}$ and $R_F = \min_{\sqsubseteq_{\mathcal{L}}}\{c(\varphi) \mid \varphi \in F\}$ its representation, $\bigwedge R_F \equiv \bigwedge F$ and $\uparrow R_F = \uparrow F$.*

Corollary 2. *If $F \subseteq \mathcal{L}$ is upward closed w.r.t. $\sqsubseteq_{\mathcal{L}}$ then $F = \uparrow R_F$.*

In particular, Corollary 2 applies to any set that is closed under entailment.

4 The Weaken Operator

This section develops an algorithm that computes a *weaken* operator, which takes a representation of an upward-closed set $F \subseteq \mathcal{L}$ and a state s and computes a representation of $F \cap \alpha(\{s\}) = \{\varphi \in F \mid s \models \varphi\}$. When F is viewed as an abstract element, this operation corresponds to computing $F \sqcup \alpha(\{s\})$. While it is not a general abstract join operator, joining an abstract element with the abstraction of a single concrete state is a powerful building block that can be used, for example, to compute the abstraction of a set of states or even the least fixpoint of the best abstract transformer (*á la* symbolic abstraction [27]).

In an explicit representation of F, computing $F \sqcup \alpha(\{s\})$ would amount to removing from F all the formulas that are not satisfied by s. However, in the subsumption-based representation R_F, simply removing said formulas is not enough. Instead, we must *weaken* them, i.e., replace them by formulas they subsume that are satisfied by s. To this end, Sect. 4.1 develops an appropriate weakening operator for a single formula, and Sect. 4.2 then lifts it to antichains used as representations.

4.1 Weakening a Single Canonical Formula

Given a canonical formula φ and a state s such that $s \not\models \varphi$, the weaken operator computes the set of minimal canonical formulas that are subsumed by φ and satisfied by s, which can be understood as a representation of $\uparrow\{\varphi\} \cap \alpha(\{s\})$.

Definition 5 (The Weaken Operator). *The* weaken operator *of $\mathcal{L}$ is the function $\mathcal{W}_{\mathcal{L}}\colon \boldsymbol{\mathcal{L}} \times \mathbb{S} \to \mathcal{P}(\boldsymbol{\mathcal{L}})$ defined as follows:*

$$\mathcal{W}_{\mathcal{L}}(\varphi, s) = \min_{\sqsubseteq_{\mathcal{L}}} \{c_{\mathcal{L}}(\psi) \mid \psi \in \mathcal{L}, \varphi \sqsubseteq \psi, \text{ and } s \models \psi\}.$$

Note that $\mathcal{W}_{\mathcal{L}}(\varphi, s)$ returns a set of formulas, since there may be different incomparable ways to weaken φ such that it is satisfied by s.

While Definition 5 does not suggest a way to compute $\mathcal{W}_{\mathcal{L}}(\varphi, s)$, the following theorem provides a recursive implementation of $\mathcal{W}_{\mathcal{L}}(\varphi, s)$ that follows the inductive structure of bounded languages. For the quantification cases, we weaken according to all assignments of variables in $X \subseteq V$. Recall that a state can be unpacked as $s = ((\mathcal{U}, \mathcal{I}), \mu)$ where $(\mathcal{U}, \mathcal{I})$ is a first-order structure (universe and interpretation) and μ is an assignment to variables (into $\mathcal{U}$). For assignments μ and ν, we use $\mu \overleftarrow{\cup} \nu$ to denote the assignment obtained from μ by updating (possibly extending) it according to ν.

Theorem 4 (Implementation of Weaken). *Let $\varphi \in \mathcal{L}$ be a canonical formula in a bounded first-order language $\mathcal{L}$ and $s \in \mathbb{S}$ a state. If $s \models \varphi$ then $\mathcal{W}_{\mathcal{L}}(\varphi, s) = \{\varphi\}$. If $s \not\models \varphi$, then $\mathcal{W}_{\mathcal{L}}(\varphi, s)$ is given by:*

$$
\begin{aligned}
\mathcal{W}_{\mathcal{L}_A}(\varphi, s) &= \begin{cases} \{\psi \in A \mid s \models \psi\}, & \textit{if } \varphi = \bot \\ \emptyset, & \textit{if } \varphi \neq \bot \end{cases} \\
\mathcal{W}_{\vee[\mathcal{L}_1,\mathcal{L}_2]}(\varphi_1 \vee \varphi_2, s) &= \{\psi \vee \varphi_2 \mid \psi \in \mathcal{W}_{\mathcal{L}_1}(\varphi_1, s)\} \cup \{\varphi_1 \vee \psi \mid \psi \in \mathcal{W}_{\mathcal{L}_2}(\varphi_2, s)\} \\
\mathcal{W}_{\wedge[\mathcal{L}_1,\mathcal{L}_2]}(\varphi_1 \wedge \varphi_2, s) &= \{\psi_1 \wedge \psi_2 \mid \psi_1 \in \mathcal{W}_{\mathcal{L}_1}(\varphi_1, s), \psi_2 \in \mathcal{W}_{\mathcal{L}_2}(\varphi_2, s)\} \\
\mathcal{W}_{\vee_k[\mathcal{L}]}\left(\bigvee \bar{\varphi}, s\right) &= \min{}_{\sqsubseteq_{\vee_k[\mathcal{L}]}}\left(W_{|\bar{\varphi}|} \cup W_{|\bar{\varphi}|+1}\right) \;\; \textit{where} \\
W_{|\bar{\varphi}|} &= \left\{\bigvee \langle \varphi_1, \ldots, \varphi_{i-1}, \psi, \varphi_{i+1}, \ldots, \varphi_{|\bar{\varphi}|}\rangle_{\leq_{\mathcal{L}}} \mid i \in [\bar{\varphi}], \psi \in \mathcal{W}_{\mathcal{L}}(\varphi_i, s)\right\} \;\; \textit{and} \\
W_{|\bar{\varphi}|+1} &= \left\{\bigvee \langle \varphi_1, \ldots, \varphi_{|\bar{\varphi}|}, \psi\rangle_{\leq_{\mathcal{L}}} \mid \psi \in \mathcal{W}_{\mathcal{L}}(\bot_{\mathcal{L}}, s) \;\; \textit{and} \; |\bar{\varphi}| < k\right\} \\
\mathcal{W}_{\wedge_\omega[\mathcal{L}]}\left(\bigwedge \bar{\varphi}, s\right) &= \left\{\bigwedge \langle \min{}_{\sqsubseteq_{\mathcal{L}}} \mathcal{W}_{\mathcal{L}}(\varphi_1, s) \cup \cdots \cup \mathcal{W}_{\mathcal{L}}\left(\varphi_{|\bar{\varphi}|}, s\right)\rangle_{\leq_{\mathcal{L}}}\right\} \\
\mathcal{W}_{\exists_X[\mathcal{L}]}(\exists X.\varphi, ((\mathcal{U}, \mathcal{I}), \mu)) &= \min{}_{\sqsubseteq_{\exists_X[\mathcal{L}]}}\{c(\exists X.\psi) \mid \nu\colon X \to \mathcal{U}, \psi \in \mathcal{W}_{\mathcal{L}}\left(\varphi, ((\mathcal{U}, \mathcal{I}), \mu \overset{\leftarrow}{\cup} \nu)\right)\} \\
\mathcal{W}_{\forall_X[\mathcal{L}]}(\forall X.\varphi, ((\mathcal{U}, \mathcal{I}), \mu)) &= \min{}_{\sqsubseteq_{\forall_X[\mathcal{L}]}}\{c(\forall X.\psi) \mid \psi \in \Omega_{\varphi}\left(\{((\mathcal{U}, \mathcal{I}), \mu \overset{\leftarrow}{\cup} \nu) \mid \nu\colon X \to \mathcal{U}\}\right)\} \\
\textit{where } \Omega_{\varphi_0}(\{s_1, \ldots, s_n\}) &= \{\varphi_n \mid \varphi_1 \in \mathcal{W}_{\mathcal{L}}(\varphi_0, s_1), \ldots, \varphi_n \in \mathcal{W}_{\mathcal{L}}(\varphi_{n-1}, s_n)\} \\
\mathcal{W}_{\exists\!\!\forall_X[\mathcal{L}]}(\exists X.\varphi, s) &= \mathcal{W}_{\exists_X[\mathcal{L}]}(\exists X.\varphi, s) \\
\mathcal{W}_{\exists\!\!\forall_X[\mathcal{L}]}(\forall X.\varphi, s) &= \min{}_{\sqsubseteq_{\exists\!\!\forall_X[\mathcal{L}]}}\left(\mathcal{W}_{\exists_X[\mathcal{L}]}(\exists X.\varphi, s) \cup \mathcal{W}_{\forall_X[\mathcal{L}]}(\forall X.\varphi, s)\right)
\end{aligned}
$$

When $s \models \varphi$, no weakening of φ is needed for s to satisfy it. In the case of $\mathcal{L}_A$, only $\bot$ can be weakened to make s satisfy it, yielding the set of formulas from A that are satisfied by s. (For $\mathcal{L}_A$, weakening anything except $\bot$ that is not satisfied by s yields the empty set.) In the case of disjunction, it suffices for one of the disjuncts to be satisfied by s. Therefore, weakening is done by (i) weakening exactly one of the existing disjuncts, which applies to both $\vee[\mathcal{L}_1, \mathcal{L}_2]$ and $\vee_k[\mathcal{L}]$; or by (ii) adding a disjunct that weakens $\bot_{\mathcal{L}}$, which applies only to $\bigvee \bar{\varphi} \in \vee_k[\mathcal{L}]$ when $|\bar{\varphi}| < k$. In the case of homogeneous disjunction, each resulting disjunction needs to be sorted to restore canonicity; moreover, some of the resulting disjunctions may be subsumed by others, so $\min_{\sqsubseteq_{\vee_k[\mathcal{L}]}}$ is applied to the set of weakened disjunctions. In the case of conjunction, all conjuncts need to be weakened to be satisfied by s. In the binary case, this leads to all pairs that combine weakened conjuncts. But in the homogeneous case a single conjunction can accumulate all weakened conjuncts, so weakening always yields a singleton set; filtering the weakened conjuncts using $\min_{\sqsubseteq_{\mathcal{L}}}$ is required to ensure canonicity, as one weakened conjunct may subsume another. To satisfy an existentially quantified formula, it suffices for the body to be satisfied by a single assignment. Therefore, each possible assignment ν contributes to the result of weakening. In contrast, for a universally quantified formula the body must be satisfied by all assignments. Therefore, the body of the formula is iteratively weakened by *all* assignments. In both cases, formulas are re-canonicalized and non-minimal elements are removed. The case of $\exists\!\!\forall_X[\mathcal{L}]$ combines the two quantified cases.

Example 2. Consider applying the weaken operator of $\mathcal{L} = \forall_{\{x,y\}}[\vee_2[\mathcal{L}_A]]$ from Example 1 to the bottom element $\bot_{\mathcal{L}} = \forall\{x, y\}. \bigvee \epsilon$, with the state $s = ((\mathcal{U}, \mathcal{I}), \mu)$ where $\mathcal{U} = \{a, b\}$, $p^{\mathcal{I}} = \{a, b\}$, and μ is an empty assignment. To weaken the universally quantified formula, we first iteratively weaken its body,

$\varphi_0 = \bigvee \epsilon$, with the states $s_1, \ldots, s_4$, each of which extends s with one of the 4 possible assignments to x, y. Since all of these states satisfy $p(x)$ and $p(y)$, the first weakening (with s_1) results in $\{\bigvee\langle p(x)\rangle, \bigvee\langle p(y)\rangle\}$, and no formula is weakened further in later iterations (since both of them are already satisfied by s_2, s_3, s_4). As we have seen in Sect. 3.3, both formulas are canonical; however, they become subsumption-equivalent when the quantifier prefix is added, demonstrating the need for additional canonicalization in the computation of weaken for $\forall_X[\cdot]$. The result is the antichain of canonical formulas $\{\forall\{x, y\}. \bigvee\langle p(x)\rangle\}$. Note that the weakened formula $\bot_{\mathcal{L}}$ has 21 formulas in its $\sqsubseteq_{\mathcal{L}}$-upward closure, and its weakening has 14 formulas (see [8] for the lists of formulas); yet throughout the weakening process we only dealt with at most two formulas.

Algorithm 1: In-place Weaken for LSet[$\mathcal{L}$]

```
Input: An antichain of canonical $\mathcal{L}$-formulas $R$ stored in the LSet[$\mathcal{L}$] data
       structure and a state $s \in \mathbb{S}$
Output: $R$ modified in place to store $\mathcal{W}_{\mathcal{L}}(R, s)$
$U := R|_{\not\models s}$;
for $\varphi \in U$ do $R$.remove($\varphi$);
$W := \bigcup_{\varphi \in U} \mathcal{W}_{\mathcal{L}}(\varphi, s)$;
for $\varphi \in W$ sorted by $\leq_{\mathcal{L}}$ do
    if $R|_{\sqsubseteq \varphi} = \emptyset$ then $R$.insert($\varphi$);
```

4.2 Weakening Sets of Formulas

We lift the weaken operator to sets of canonical formulas. For a set $R \subseteq \boldsymbol{\mathcal{L}}$, we define $\mathcal{W}_{\mathcal{L}}(R, s) = \min_{\sqsubseteq} \bigcup_{\varphi \in R} \mathcal{W}_{\mathcal{L}}(\varphi, s)$, motivated by the following theorem.

Theorem 5 (From Weaken to Join). *Let $F \subseteq \mathcal{L}$ be upward-closed w.r.t. $\sqsubseteq$, R_F its representation ($R_F = \min_{\sqsubseteq}\{c(\varphi) \mid \varphi \in F\}$), and s a state. The representation of $F \sqcup \alpha(\{s\})$ is given by $\mathcal{W}_{\mathcal{L}}(R_F, s) = \min_{\sqsubseteq} \bigcup_{\varphi \in R_F} \mathcal{W}_{\mathcal{L}}(\varphi, s)$.*

Corollary 3 (Weaken for a Set of States). *Let $F \subseteq \mathcal{L}$ be upward-closed w.r.t. $\sqsubseteq$, R_F its representation, and $s_1, \ldots, s_n$ states. The representation of $F \sqcup \alpha(\{s_1, \ldots, s_n\})$ is given by $\mathcal{W}_{\mathcal{L}}(\mathcal{W}_{\mathcal{L}}(\cdots \mathcal{W}_{\mathcal{L}}(\mathcal{W}_{\mathcal{L}}(R_F, s_1), s_2), \cdots s_{n-1}), s_n)$.*

Corollary 4 (Abstraction of a Set of States). *The representation of $\alpha(S)$ for a set of states S The representation of $\alpha(\{s_1, \ldots, s_n\})$ is given by $\mathcal{W}_{\mathcal{L}}(\mathcal{W}_{\mathcal{L}}(\cdots \mathcal{W}_{\mathcal{L}}(\mathcal{W}_{\mathcal{L}}(\{\bot_{\mathcal{L}}\}, s_1), s_2), \cdots s_{n-1}), s_n)$.*

Theorem 5 and Corollary 3 show that weakening of a single formula can be lifted to compute join between an upward-closed set of formulas (represented using its minimal elements w.r.t. $\sqsubseteq$) and the abstraction of one or more states.

Next, we observe that we can implement $\mathcal{W}_{\mathcal{L}}(R, s)$ by (i) focusing only on formulas that actually need weakening, i.e., formulas in R that are not satisfied by s, without iterating over formulas that s satisfies; and (ii) leveraging the $\leq_{\mathcal{L}}$ total order to accumulate the set of minimal elements more efficiently.

Algorithm 1 presents our implementation of $\mathcal{W}_{\mathcal{L}}(R, s)$ for an antichain R of canonical formulas and a state s. It updates R to $\mathcal{W}_{\mathcal{L}}(R, s)$ in place, which is useful for computing an abstraction of a set of states (Corollary 3) or even for fixpoint computation (Sect. 6). The algorithm uses a data structure LSet[$\mathcal{L}$] (whose implementation is explained in Sect. 5) that stores a set of canonical $\mathcal{L}$-formulas and supports two efficient filters: one for formulas that are not satisfied by a given state s, denoted by $R|_{\not\models s}$; and one for formulas that subsume a given formula φ, denoted by $R|_{\sqsubseteq\varphi}$. Formally: $R|_{\not\models s} = \{\psi \in R \mid s \not\models \psi\}$ and $R|_{\sqsubseteq\varphi} = \{\psi \in R \mid \varphi \sqsupseteq \psi\}$.

To weaken R, Algorithm 1 first identifies all formulas that need weakening using the $R|_{\not\models s}$ filter. It then removes these formulas, weakens them, and adds the weakened formulas back to the set, while filtering out formulas that are not $\sqsubseteq_{\mathcal{L}}$-minimal. For the minimality filtering, we leverage $\leq_{\mathcal{L}}$ to ensure that if $\varphi \sqsubseteq_{\mathcal{L}} \psi$ then φ is added before ψ. As a result, when inserting a formula φ we only need to check that it is not already subsumed by another formula in the set, which is done by checking if $R|_{\sqsubseteq\varphi}$ is empty[3]. Importantly, a formula $\varphi \in R \setminus R|_{\not\models s}$ cannot be subsumed by a formula from $\mathcal{W}_{\mathcal{L}}(\psi, s)$ for $\psi \in R|_{\not\models s}$. (If we assume the contrary we easily get that $\psi \sqsubseteq \varphi$, contradicting the fact that R is an antichain.)

4.3 Design Consideration and Tradeoffs

We are now in a position to discuss the tradeoffs and considerations that arise in our framework in the design of languages and their subsumption relations, explaining the design choices behind Definitions 1 and 2.

There is a tradeoff between the precision of the subsumption relation $\sqsubseteq_{\mathcal{L}}$ and the complexity of implementing the weaken operator $\mathcal{W}_{\mathcal{L}}$. From a representation perspective, a more precise $\sqsubseteq_{\mathcal{L}}$ is desirable (i.e., relating more formulas), since it means that the upward closure $\uparrow\{\varphi\}$ of a formula φ is larger, and (upward-closed) sets of formulas can be represented using less minimal formulas. On the other hand, when $\uparrow\{\varphi\}$ is larger, computing $\mathcal{W}_{\mathcal{L}}(\varphi, s)$ is generally more complicated. As an extreme case, if $\sqsubseteq_{\mathcal{L}}$ is trivial (i.e., a formula only subsumes itself), we get no pruning in the representation, but computing $\mathcal{W}_{\mathcal{L}}(\varphi, s)$ is very easy, since it is either $\{\varphi\}$ or $\emptyset$. As another example, compare $\vee[\mathcal{L}, \mathcal{L}]$ with $\vee_2[\mathcal{L}]$. The subsumption relation of $\vee[\mathcal{L}, \mathcal{L}]$ is a pointwise extension, while that of $\vee_2[\mathcal{L}]$ allows swapping the two formulas, which is more precise. (E.g., $\bigvee\langle\varphi, \psi\rangle \sqsubseteq_{\vee_2[\mathcal{L}]} \bigvee\langle\psi, \varphi\rangle$ always holds but we might have $\varphi \vee \psi \not\sqsubseteq_{\vee[\mathcal{L},\mathcal{L}]} \psi \vee \varphi$.) Accordingly, weakening of $\vee_2[\mathcal{L}]$-formulas is slightly more involved.

As opposed to reordering of disjuncts, $\sqsubseteq_{\vee_k[\mathcal{L}]}$ does not allow multiple disjuncts to subsume the same one, e.g., $\bigvee\langle\varphi, \psi\rangle \not\sqsubseteq_{\vee_k[\mathcal{L}]} \bigvee\langle\psi\rangle$ even if $\varphi \sqsubseteq_{\mathcal{L}} \psi$ (recall that the mapping between disjuncts must be injective). This choice makes the

[3] While the implementation of the weaken operator only checks the emptiness of $R|_{\sqsubseteq\varphi}$, the full set is used in the recursive implementation of $R|_{\sqsubseteq\varphi}$ (Sect. 5).

computation of $\mathcal{W}_{\vee_k[\mathcal{L}]}$ simpler, as it only needs to consider individually weakening each disjunct or adding a new one, but not merging of disjuncts (to "make space" for a new disjunct). For example, when computing $\mathcal{W}_{\vee_2[\mathcal{L}]}\left(\bigvee\langle\varphi_1,\varphi_2\rangle, s\right)$, we do not have to consider formulas of the form $\bigvee\langle\varphi,\psi\rangle$ where $s \models \psi$ and $\varphi_1,\varphi_2 \sqsubseteq_{\mathcal{L}} \varphi$, which we would need to include if the mapping was not required to be injective. One seemingly undesirable consequence of the injectivity requirement is that canonical formulas may contain redundant disjuncts, e.g., $\bigvee\langle\varphi,\psi\rangle$ when $\varphi \sqsubseteq \psi$ (or even $\bigvee\langle\varphi,\varphi\rangle$). However, when formulas are obtained by iterative weakening, as in the computation of the representation of $\alpha(S)$ for a set of concrete states S, formulas with such redundancies will be eliminated as they are always subsumed by a canonical formula without redundancies.

Our design of bounded first-order languages uses bounded disjunction but unbounded conjunction. The reason is that we obtain formulas by weakening other formulas, starting from $\bot_{\mathcal{L}}$. In this scenario, bounding the size of conjunctions would have replaced one conjunction by all of its subsets smaller than the bound, causing an exponential blowup in the number of formulas, without contributing much to generalization. On the other hand, bounding the size of disjunctions yields generalization without blowing up the number of formulas (in fact, it reduces the number of formulas compared to unbounded disjunction).

5 Data Structure for Sets of Formulas

The implementation of $\mathcal{W}_{\mathcal{L}}\left(R, s\right)$ presented in Algorithm 1 uses the filters $R|_{\not\models s}$ and $R|_{\sqsubseteq\varphi}$. Since the sets may be very large, a naive implementation that iterates over R to find formulas that are not satisfied by s ($R|_{\not\models s}$) or formulas that subsume φ ($R|_{\sqsubseteq\varphi}$) may become inefficient. We therefore introduce a data structure for bounded first-order languages, which we call LSet[$\mathcal{L}$], that stores a set of canonical $\mathcal{L}$-formulas R (not necessarily an antichain), and implements $R|_{\not\models s}$ and $R|_{\sqsubseteq\varphi}$ without iterating over all formulas in R. The key idea is to define the LSet[$\mathcal{L}$] data structure recursively, following the structure of $\mathcal{L}$, and to use auxiliary data to implement the $R|_{\not\models s}$ and $R|_{\sqsubseteq\varphi}$ filters more efficiently.

For example, to implement LSet[$\vee[\mathcal{L}_1,\mathcal{L}_2]$], we store a set of $\vee[\mathcal{L}_1,\mathcal{L}_2]$-formulas and two auxiliary data fields: an LSet L : LSet[$\mathcal{L}_1$] and a map M : Map[$\mathcal{L}_1$, LSet[$\mathcal{L}_2$]]. We maintain the invariant that $\varphi_1 \vee \varphi_2$ is in the set iff $\varphi_2 \in M[\varphi_1]$, and that L contains the same $\mathcal{L}_1$-formulas as the keys of M. Then, to find formulas that are not satisfied by a state s, i.e., formulas where both disjuncts are not satisfied by s, we first query L to find φ_1's that are not satisfied by s, and for each such φ_1 we query the LSet $M[\varphi_1]$ to find φ_2's that are not satisfied by s. Implementing the subsumption filter follows a similar logic.

Our implementation of LSet[$\vee_k[\mathcal{L}]$] uses a trie data structure that generalizes the binary case. Each edge is labeled by an $\mathcal{L}$-formula, and each node represents an $\vee_k[\mathcal{L}]$-formula that is the disjunction of the edge labels along the path from the root to the node. The outgoing edges of each node are stored using an LSet[$\mathcal{L}$] that can be used to filter only the edges whose label is not satisfied by a given state, or subsumes a given formula. Then, the $R|_{\not\models s}$ and $R|_{\sqsubseteq\bigvee\bar{\varphi}}$ filters are implemented by recursive traversals of the tree that only traverse filtered edges.

The recursive implementation for the other language constructors is simpler, and follows a similar intuition to that of the cases presented above. The base case $\mathrm{LSet}[\mathcal{L}_A]$ is implemented without any auxiliary data using straightforward iteration. The full details of the $\mathrm{LSet}[\mathcal{L}]$ data structure appear in [8].

6 Implementation and Evaluation

To evaluate our abstract domain implementation, we used it to implement a symbolic abstraction [27,29] algorithm that computes the least fixpoint of the best abstract transformer of a transition system. We evaluated our implementation on 19 distributed protocols commonly used as benchmarks in safety verification and obtained promising results.

6.1 Implementation

We implemented our abstract domain and the symbolic abstraction algorithm in Flyvy,[4] an open-source verification tool written in Rust, whose implementation leverages parallelism and the optimizations detailed below. The implementation and benchmarks used, as well as the log files and raw results from the experiments reported, are publicly available in this paper's artifact [7].

Our implementation receives as input (i) a first-order transition system (ι, τ) over signature Σ, where ι is a closed first-order formula over Σ specifying the initial states and τ is a closed first-order formula over two copies of Σ specifying the transitions, and (ii) a specification of a bounded first-order language $\mathcal{L}$ over Σ that defines the abstract domain $\mathcal{P}(\mathcal{L})$. The reachable states of the system are the least fixpoint of a concrete transformer $\mathcal{T} : \mathcal{P}(\mathbb{S}) \to \mathcal{P}(\mathbb{S})$ given by $\mathcal{T}(S) = \{s' \in \mathbb{S} \mid s' \models \iota \vee \exists s \in S.\ \langle s, s' \rangle \models \tau\}$, where $\langle s, s' \rangle \models \tau$ indicates that the pair of states satisfies the two-vocabulary formula τ, i.e., that s' is a successor of s w.r.t the transition relation defined by τ. For more details on this style of modeling distributed systems in first-order logic, see [23–25].

The Galois connection (α, γ) between $\mathcal{P}(\mathbb{S})$ and $\mathcal{P}(\mathcal{L})$ induces a *best abstract transformer* $\mathcal{T}^\sharp : \mathcal{P}(\mathcal{L}) \to \mathcal{P}(\mathcal{L})$ defined by $\mathcal{T}^\sharp = \alpha \circ \mathcal{T} \circ \gamma$. Any fixpoint of $\mathcal{T}^\sharp$, i.e., a set $F \subseteq \mathcal{L}$ such that $\mathcal{T}^\sharp(F) = F$, is an *inductive invariant* of (ι, τ) (when sets are interpreted conjunctively), and the least fixpoint, $\mathrm{lfp}\mathcal{T}^\sharp$, is the strongest inductive invariant in $\mathcal{L}$. The strongest inductive invariant is useful for verifying safety properties of the system, or showing that they cannot be proven in $\mathcal{L}$ (if the strongest inductive invariant in $\mathcal{L}$ cannot prove safety, neither can any other inductive invariant expressible in $\mathcal{L}$).

Symbolic abstraction computes $\mathrm{lfp}\mathcal{T}^\sharp$ without computing $\mathcal{T}^\sharp$ explicitly: beginning with $F = \mathcal{L}$ (the least element in $\mathcal{P}(\mathcal{L})$), and as long as $F \neq \mathcal{T}^\sharp(F)$, a *counterexample to induction* (CTI) of F is sampled, i.e., a state $s' \not\models \bigwedge F$ that is either an initial state or the successor of a state s with $s \models \bigwedge F$, and F is updated to $F \sqcup \alpha(\{s'\})$. Our implementation uses the representation R_F and

[4] Flyvy's code is available at https://github.com/vmware-research/temporal-verifier.

Algorithm 1 to compute the join (more details in [8]). To find CTIs or determine that none exist we use SMT solvers (Z3 [21] and cvc5 [2]), with queries restricted to the EPR fragment (following [24]), which ensures decidability and the existence of finite counterexamples. Solvers still struggle in some challenging benchmarks, and we employ several optimizations detailed in [8] to avoid solver timeouts.

6.2 Experiments

To evaluate our techniques, we computed the least fixpoints (strongest inductive invariants) of 19 distributed protocols commonly used as benchmarks in safety verification, in a language expressive enough to capture their human-written safety invariants. We used all EPR benchmarks from [15], except for universally quantified Paxos variants. To evaluate the utility of the LSet data structure described in Sect. 5, we ran each experiment twice, once using LSet and once using a naive (but parallelized) implementation for the filters $R|_{\not\Dashv s}$ and $R|_{\sqsubseteq\varphi}$.

To specify the bounded first-order language for each example, we provide the tool with a quantifier prefix (using $\exists_X[\cdot]$, $\forall_X[\cdot]$, and $\exists\!\!\!\forall_X[\cdot]$) composed on top of a quantifier-free bounded language that captures k-pDNF (following [15]). A k-pDNF formula has the structure $c_1 \to (c_2 \vee \cdots \vee c_k)$, where $c_1, c_2, \ldots, c_k$ are cubes (conjunctions of literals). We specify such formulas as $\bigvee[\bigvee_n[\mathcal{L}_{A_1}], \bigvee_{k-1}[\bigwedge_\omega[\mathcal{L}_{A_2}]]]$, where k and n are parameters, and A_1 and A_2 are sets of literals. Inspired by [30], we observe that we can restrict the variables used in A_1 and A_2 to reduce the size of the language without losing precision.[5] For additional details see [8]. The list of examples with their language parameters appears in Table 1. For each example, we report the quantifier structure, the k and n parameters of the k-pDNF quantifier-free matrix, and the approximate size of the language $\mathcal{L}$. Recall that the size of the abstract domain is $2^{|\mathcal{L}|}$.

All experiments were performed on a 48-threaded machine with 384 GiB of RAM (AWS's `z1d.metal`) and a three-hour time limit. For each example we also provide runtimes of two state-of-the-art safety verification tools, DuoAI [30] and P-FOL-IC3 [15]. Note that, unlike our technique, these tools look for *some* inductive invariant proving safety, not necessarily the strongest, but are also given fewer explicit language constraints. Moreover, the runtimes of DuoAI and P-FOL-IC3 are sourced from their respective papers, and reflect different architectures and time limits. Thus, the inclusion of their results is not intended as a precise comparison to our tool, but as a reference for the difficulty of the invariant inference task of each example, as evidenced by state-of-the-art techniques.

[5] One of the language reductions used by [30] relies on an overly generalized lemma [30, Lemma 6]; we confirmed this with the authors of [30]. We prove and use a correct (but less general) variant of this lemma, see [8] for details.

6.3 Results

The results of the symbolic abstraction computation are presented in Table 1. For each experiment we report the runtime of our tool and the following statistics: the percentage of time spent weakening formulas (as opposed to searching for CTIs), the number of formulas in the representation of the fixpoint (if reached), and the maximal number of formulas in the representation of an abstract element throughout the run. Each experiment was run five times, unless it timed out, in which case it was run only once. We aggregate the results of each statistic across multiple runs as *median* ± *deviation*, where *deviation* is the maximal distance between the median value and the value of the statistic in any given run.

For simple examples, the fixpoint computation terminates very quickly, often faster than the other tools, and maintains only tens or hundreds of formulas throughout its run. Some of the larger examples, such as `ticket`, `paxos-epr`, `flexible-paxos-epr`, and `cache` also terminate after similar times to the other tools. In fact, this is the first work to compute least fixpoints for any Paxos variant or `cache`. (DuoAI, for instance, has a component that attempts to compute a precise fixpoint, but [30] reports that it times out on all Paxos variants.)

Unsurprisingly, there is a significant gap between the runtimes of examples with and without quantifier alternation, mostly due to the time spent in SMT solvers. For example, in `ticket` we spend about 43% of the runtime performing weakenings, but this percentage drops to 1% and 4% for `paxos-epr` and `flexible-paxos-epr`, respectively. This causes the runtime of `paxos-epr` to exceed that of `ticket` by more than an order of magnitude, although its fixpoint computation considers fewer formulas and actually spends less time weakening. Similarly, in `cache` we manage to prove a fixpoint of a hundred thousand formulas in about an hour and spend a third of it weakening formulas, while `multi-paxos-epr` and `fast-paxos-epr` time out, although they consider far fewer formulas and spend a negligible amount of time weakening.

Next, we observe that the use of LSet significantly reduces time spent in weakening, leading to more than an order of magnitude difference even in moderate examples, e.g., `ticket` and `paxos-epr`. In terms of the *total* fixpoint computation time, in examples where the runtime is small or dominated by the SMT solvers, the effect might be negligible, but otherwise the speedup is significant. For example, `cache` is not solved within the 3-hour limit with a naive data structure; it gets stuck after reaching $\sim$ 20,000 formulas in the abstraction, whereas using LSet it is solved in about an hour while handling more than ten times the number of formulas. Similarly, in the two unsolved examples where SMT calls seem to be the bottleneck (`multi-paxos-epr` and `fast-paxos-epr`), using a naive data structure causes weakening to become the bottleneck and time out.

Finally, the remaining timeouts, `learning-switch`, `stoppable-paxos-epr`, and `vertical-paxos-epr`, are the only examples where the weakening process itself is the bottleneck. These are cases where the language induced by the human-written invariant, using the constraining parameters of bounded languages, create a inefficient weakening process. The cause for this is either a profusion of literals in the basis language (>600 in `learning-switch` and

Table 1. Symbolic abstraction over invariant inference benchmarks with a time limit of 3 h (10800 s). We describe the bounded language underlying the abstract domain of each example, including its approximate size, and report the runtime of our technique—with and without using LSet—along with some statistics. For reference, we provide runtimes of two state-of-the-art safety-verification tools. 'T/O' indicates a timeout, and 'N/A' indicates that the example was not reported by the respective tool.

Example	Language				Runtime (sec)	LSet	% in $\mathcal{W}$	Lfp. Size	Max. Size	Safety (sec)	
	quant	k	n	size						P-FOL-IC3	DuoAI
lockserv	$\forall^2$	1	3	10^4	**0.4 ± 0.1**	✓	6 ± 1 %	12	28 ± 7	19	1.9
					0.5 ± 0.1	–	6 ± 1 %		29 ± 3		
toy-consensus-forall	$\forall^3$	1	3	10^3	**0.2 ± 0.0**	✓	9 ± 2 %	5	18 ± 5	4	1.9
					0.2 ± 0.0	–	7 ± 1 %		18 ± 4		
ring-id	$\forall^3$	1	3	10^5	**1.6 ± 0.1**	✓	16 ± 1 %	97	182 ± 22	7	3.5
					1.9 ± 0.1	–	20 ± 1 %		189 ± 22		
sharded-kv	$\forall^5$	1	3	10^4	**0.5 ± 0.1**	✓	8 ± 0 %	20	26 ± 2	8	1.9
					0.5 ± 0.0	–	8 ± 1 %		26 ± 4		
ticket	$\forall^4$	1	5	10^9	**32.6 ± 3.3**	✓	43 ± 7 %	2621	8531 ± 119	23	23.9
					862.2 ± 21.9	–	97 ± 0 %		8533 ± 121		
learning-switch	$\forall^4$	1	4	10^{11}	**T/O**	✓	98 %	–	9576194	76	52.4
					T/O	–	100 %		5998		
consensus-wo-decide	$\forall^3$	1	3	10^6	**3.0 ± 0.2**	✓	19 ± 1 %	41	717 ± 109	50	3.9
					4.0 ± 0.6	–	38 ± 4 %		724 ± 43		
consensus-forall	$\forall^4$	1	3	10^6	**3.5 ± 0.4**	✓	21 ± 2 %	51	740 ± 114	1980	11.9
					5.1 ± 0.9	–	40 ± 6 %		708 ± 82		
cache	$\forall^6$	1	5	10^{11}	**4029.4 ± 220.7**	✓	30 ± 2 %	106348	271255 ± 13081	2492	N/A
					T/O	–	100 ± 0 %		19183 ± 9466		
sharded-kv-no-lost-keys	$\forall^1(\exists\forall)^2$	1	2	10^2	**0.3 ± 0.0**	✓	3 ± 0 %	4	4 ± 0	4	2.1
					0.3 ± 0.0	–	2 ± 0 %		4 ± 0		
toy-consensus-epr	$\forall^2(\exists\forall)^1\forall^1$	1	3	10^4	**0.3 ± 0.0**	✓	9 ± 1 %	5	18 ± 4	4	2.6
					0.3 ± 0.0	–	7 ± 1 %		19 ± 3		
consensus-epr	$(\exists\forall)^1\forall^4$	1	3	10^6	**5.1 ± 0.7**	✓	17 ± 2 %	51	800 ± 137	37	4.8
					8.8 ± 1.7	–	46 ± 8 %		783 ± 88		
client-server-ae	$\forall^2(\exists\forall)^1$	2	1	10^3	**0.2 ± 0.0**	✓	4 ± 1 %	2	5 ± 0	4	1.5
					0.2 ± 0.1	–	3 ± 1 %		5 ± 0		
paxos-epr	$\forall^4(\exists\forall)^2$	2	3	10^{11}	**621.5 ± 246.8**	✓	1 ± 1 %	1438	1693 ± 203	920	60.4
					789.6 ± 285.5	–	11 ± 3 %		1737 ± 168		
flexible-paxos-epr	$\forall^4(\exists\forall)^2$	2	3	10^{11}	**166.7 ± 29.3**	✓	4 ± 1 %	964	1622 ± 177	418	78.7
					235.6 ± 31.7	–	35 ± 8 %		1575 ± 196		
multi-paxos-epr	$\forall^5(\exists\forall)^3$	2	3	10^{30}	**T/O**	✓	2 %	–	27508	4272	1549
					T/O	–	100 %		6400		
fast-paxos-epr	$\forall^4(\exists\forall)^3$	2	4	10^{14}	**T/O**	✓	1 %	–	16290	9630	26979
					T/O	–	99 %		13683		
stoppable-paxos-epr	$\forall^7(\exists\forall)^3$	2	5	10^{155}	**T/O**	✓	100 %	–	37529	>18297	4051
					T/O	–	100 %		3331		
vertical-paxos-epr	$\forall^4(\exists\forall)^3$	3	5	10^{54}	**T/O**	✓	100 %	–	112990	T/O	T/O
					T/O	–	100 %		2576		

`stoppable-paxos-epr`, less than 200 in all other examples), or a very expressive language (e.g., `vertical-paxos-epr` uses 3-pDNF, whereas all other examples use 1- and 2-pDNF). For these examples, it might be necessary to restrict the languages in additional ways, e.g., as was done in [30]. Our experience, however, is that the more significant bottleneck for computing least fixpoints for the most complicated examples is the SMT queries.

7 Related Work

Many recent works tackle invariant inference in first-order logic [9–11, 14, 15, 17, 26, 30, 31]. These works are all property-guided and employ sophisticated heuristics to guide the search for invariants. Of these works, the most closely related to ours are [30, 31]. DistAI [31] is restricted to universally quantified invariants, while DuoAI [30] infers invariants with quantifier alternations. DuoAI defines a "minimum implication graph" enumerating all formulas in a first-order logical language, whose transitive closure can be understood as a specific subsumption relation, and where replacing a node with its successors can be understood as a form of weakening. DuoAI's "top-down refinement" precisely computes the strongest invariant in the logical domain. However, this computation does not scale to complex examples such as all Paxos variants, in which case "bottom-up refinement" is used—a property-guided process that does not compute the strongest invariant. Our approach based on a generic subsumption relation is both more principled and more scalable, as it succeeds in computing the least fixpoint for some Paxos variants.

Another work concerning a least-fixpoint in a logical domain is [19], which computes the set of propositional clauses up to length k implied by a given formula, minimized by the subsumption relation $\sqsubseteq = \subseteq$; a trie-based data structure is used to maintain the formulas, weaken them, and check subsumption of a formula by the entire set. Both that data structure and LSet[$\bigvee_k[\cdot]$] bear similarity to UBTrees [12], also employed in [3], which store sets and implement filters for subsets and supersets. However, while UBTrees and LSets always maintain ordered tree paths, these are unordered in [19], which allows [19] to perform weakening directly on the data structure, whereas we need to remove the unsatisfied disjunctions, weaken, and insert them. On the other hand, this makes filtering for subsets in UBTrees and LSets more efficient. Also note that LSet is more general than both, since it supports a more general subsumption relation.

8 Conclusion

We have developed key algorithms and data structures for working with a logical abstract domain of quantified first-order formulas. Our fundamental idea is using a well-defined subsumption relation and a *weaken* operator induced by it. This idea makes the abstract domain feasible, and it is also extensible: while we explored one possible subsumption relation and its associated weaken operator, future work may explore others, representing different tradeoffs between pruning and weakening. We demonstrated the feasibility of our approach by computing the least abstract fixpoint for several distributed protocols modeled in first-order logic—a challenging application domain where previously only property-directed heuristics have been successful. For some of the examples in our evaluation, the computation still times out. In some of these cases, SMT queries (for computing CTIs) become the bottleneck. Dealing with this bottleneck is an orthogonal problem that we leave for future work. For the examples with the largest logical languages, abstract domain operations remain the bottleneck, and future work may either scale the abstract domain implementation to such languages or explore combinations with property-directed approaches.

Acknowledgments. We thank Alex Fischman and James R. Wilcox for their contributions to the Flyvy verification tool. We thank Raz Lotan, Kenneth McMillan, and the anonymous reviewers for their helpful and insightful comments.

The research leading to these results has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreement No [759102-SVIS]). This research was partially supported by the Israeli Science Foundation (ISF) grant No. 2117/23.

References

1. Ball, T., et al.: Vericon: towards verifying controller programs in software-defined networks. In: O'Boyle, M.F.P., Pingali, K. (eds.) ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2014, Edinburgh, United Kingdom, 09–11 June 2014, pp. 282–293. ACM (2014). https://doi.org/10.1145/2594291.2594317
2. Barbosa, H., et al.: cvc5: a versatile and industrial-strength SMT solver. In: TACAS 2022. LNCS, vol. 13243, pp. 415–442. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99524-9_24
3. Cadar, C., Dunbar, D., Engler, D.R., et al.: Klee: unassisted and automatic generation of high-coverage tests for complex systems programs. In: OSDI, vol. 8, pp. 209–224 (2008)
4. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Graham, R.M., Harrison, M.A., Sethi, R. (eds.) Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977, pp. 238–252. ACM (1977). https://doi.org/10.1145/512950.512973
5. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Aho, A.V., Zilles, S.N., Rosen, B.K. (eds.) Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages, San Antonio, Texas, USA, January 1979, pp. 269–282. ACM Press (1979). https://doi.org/10.1145/567752.567778
6. Feldman, Y.M.Y., Padon, O., Immerman, N., Sagiv, M., Shoham, S.: Bounded quantifier instantiation for checking inductive invariants. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 76–95. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_5

7. Frenkel, E., Chajed, T., Padon, O., Shoham, S.: Efficient implementation of an abstract domain of quantified first-order formulas (artifact) (2024). https://doi.org/10.5281/zenodo.10938367
8. Frenkel, E., Chajed, T., Padon, O., Shoham, S.: Efficient implementation of an abstract domain of quantified first-order formulas (extended version) (2024). https://doi.org/10.48550/arXiv.2405.10308
9. Goel, A., Sakallah, K.: On symmetry and quantification: a new approach to verify distributed protocols. In: Dutle, A., Moscato, M.M., Titolo, L., Muñoz, C.A., Perez, I. (eds.) NFM 2021. LNCS, vol. 12673, pp. 131–150. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-76384-8_9
10. Goel, A., Sakallah, K.A.: Towards an automatic proof of Lamport's Paxos. In: FMCAD, pp. 112–122. IEEE (2021). https://doi.org/10.34727/2021/isbn.978-3-85448-046-4_20
11. Hance, T., Heule, M., Martins, R., Parno, B.: Finding invariants of distributed systems: It's a small (enough) world after all. In: Mickens, J., Teixeira, R. (eds.) 18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, 12–14 April 2021, pp. 115–131. USENIX Association (2021). https://www.usenix.org/conference/nsdi21/presentation/hance
12. Hoffmann, J., Koehler, J.: A new method to index and query sets. In: IJCAI, vol. 99, pp. 462–467 (1999)
13. Itzhaky, S., Banerjee, A., Immerman, N., Nanevski, A., Sagiv, M.: Effectively-propositional reasoning about reachability in linked data structures. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 756–772. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_53
14. Karbyshev, A., Bjørner, N.S., Itzhaky, S., Rinetzky, N., Shoham, S.: Property-directed inference of universal invariants or proving their absence. J. ACM **64**(1), 7:1–7:33 (2017). https://doi.org/10.1145/3022187
15. Koenig, J.R., Padon, O., Shoham, S., Aiken, A.: Inferring Invariants with quantifier alternations: taming the search space explosion. In: TACAS 2022. LNCS, vol. 13243, pp. 338–356. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99524-9_18
16. Löding, C., Madhusudan, P., Peña, L.: Foundations for natural proofs and quantifier instantiation. Proc. ACM Program. Lang. **2**(POPL), 10:1–10:30 (2018). https://doi.org/10.1145/3158098
17. Ma, H., Goel, A., Jeannin, J., Kapritsos, M., Kasikci, B., Sakallah, K.A.: I4: incremental inference of inductive invariants for verification of distributed protocols. In: SOSP, pp. 370–384. ACM (2019). https://doi.org/10.1145/3341301.3359651
18. Mathur, U., Madhusudan, P., Viswanathan, M.: What's decidable about program verification modulo axioms? In: TACAS 2020. LNCS, vol. 12079, pp. 158–177. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45237-7_10
19. McMillan, K.: Don't-care computation using k-clause approximation. In: Proceedings of the IWLS 2005, pp. 153–160 (2005)
20. McMillan, K.L., Padon, O.: Deductive verification in decidable fragments with ivy. In: Podelski, A. (ed.) SAS 2018. LNCS, vol. 11002, pp. 43–55. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99725-4_4
21. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
22. Murali, A., Peña, L., Blanchard, E., Löding, C., Madhusudan, P.: Model-guided synthesis of inductive lemmas for FOL with least fixpoints. Proc. ACM Program. Lang. **6**(OOPSLA2), 1873–1902 (2022). https://doi.org/10.1145/3563354

23. Padon, O.: Deductive Verification of Distributed Protocols in First-Order Logic. Ph.D. thesis, Tel Aviv University (2019)
24. Padon, O., Losa, G., Sagiv, M., Shoham, S.: Paxos made EPR: decidable reasoning about distributed protocols. Proc. ACM Program. Lang. **1**(OOPSLA), 108:1–108:31 (2017). https://doi.org/10.1145/3140568
25. Padon, O., McMillan, K.L., Panda, A., Sagiv, M., Shoham, S.: Ivy: safety verification by interactive generalization. In: PLDI, pp. 614–630. ACM (2016). https://doi.org/10.1145/2908080.2908118
26. Padon, O., Wilcox, J.R., Koenig, J.R., McMillan, K.L., Aiken, A.: Induction duality: primal-dual search for invariants. Proc. ACM Program. Lang. **6**(POPL), 1–29 (2022). https://doi.org/10.1145/3498712
27. Reps, T., Sagiv, M., Yorsh, G.: Symbolic implementation of the best transformer. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 252–266. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24622-0_21
28. Taube, M., et al.: Modularity for decidability of deductive verification with applications to distributed systems. In: PLDI, pp. 662–677. ACM (2018). https://doi.org/10.1145/3192366.3192414
29. Thakur, A.V.: Symbolic abstraction: algorithms and applications. Ph.D. thesis, The University of Wisconsin-Madison (2014)
30. Yao, J., Tao, R., Gu, R., Nieh, J.: DuoAI: fast, automated inference of inductive invariants for verifying distributed protocols. In: Aguilera, M.K., Weatherspoon, H. (eds.) 16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, 11–13 July 2022, pp. 485–501. USENIX Association (2022). https://www.usenix.org/conference/osdi22/presentation/yao
31. Yao, J., Tao, R., Gu, R., Nieh, J., Jana, S., Ryan, G.: Distai: data-driven automated invariant learning for distributed protocols. In: OSDI, pp. 405–421. USENIX Association (2021)