Automatic Non-interference Lemmas for Parameterized Model Checking

Jesse Bingham

Abstract—*Parameterized model checking* refers to any method that extends traditional, finite-state model checking to handle systems with an arbitrary number of processes. One popular approach to this problem uses abstraction and so-called *guard strengthening*. Here a small number of processes remain intact, while the rest are abstracted away. This initially causes counter-examples, but the user can write *non-interference lemmas*, which eliminate spurious behavior by the abstracted processes. The technique is sound in that if the user writes a lemma that is not invariant, the proof will never succeed. Though the non-interference lemmas are typically much simpler than writing a full inductive invariant, there is still a non-trivial amount of human insight needed to analysis counter-examples and concoct the lemmas. In our work, we show how the process of inferring appropriate non-interference lemmas can be automated. Our approach is based on a very general theory that simply assumes a *Galois connection* between the concrete and abstract systems. Effectively, we start with the non-interference conjecture *False*, and iteratively weaken it until it is provable using the Galois connection. This produces the *strongest* non-interference lemma provable in the Galois connection. Hence, if the approach fails to prove the property, then no human lemma would help, since it is the strongest possible lemma. We instantiate this theory to a class of symmetric parameterized systems, and show how BDDs can be used to perform all involved computations. We also show how BDD-blow up that can arise when concretizing can be mitigated by using a sound over-approximation. We successfully applied the resulting tool to three parameterized verification benchmarks: the GERMAN protocol with data path, the GERMAN2004 protocol, and the FLASH protocol. To our knowledge, this is the first time automatic parameterized model checking has been done on GERMAN2004.

1 Introduction

Consider the problem of verifying a safety property of a system S with an arbitrary number n processes. One compelling method to solve this problem involves model checking a finite state system A_0 involving a small number m (typically 2 or 3) of unmodified *concrete* processes, along with an abstracted process [21], [6], [15], [18], [19], [27]. The abstracted process over-approximates the behaviors of an arbitrary number of processes. Because of this approximation, the abstracted process usually interferes with the concrete processes in some undesirable way, not possible in the unabstracted system. Hence the invariant property typically fails in A_0. However, the method allows the human to write a so-called *non-interference lemma* ψ_1 which is a conjectured invariant of S manually culled from counterexample analysis. One uses ψ_1 to restrict the abstracted process in A_0 via a technique called *guard strengthening*, which yields a new system A_1, which again may or may not satisfy the safety property. If not, the user iterates the process, writing non-interference lemmas $\psi_1, \psi_2, \ldots$ and producing a succession of abstract systems $A_0, A_1, A_2, \ldots$ that are model checked. Assuming a system A_k is found to satisfy the property, one must still confirm that $\bigwedge_i \psi_i$ is an invariant of S. Surprisingly, the theory allows one to decide this by checking that $\bigwedge_i \psi_i$ is an invariant of A_k; this is counter-intuitive because A_k is in a sense constructed under the assumption that $\bigwedge_i \psi_i$ is an invariant. Assuming that this check also passes, we have proved that the system S is safe for arbitrary n.

In this paper we automate this process. We compute the *strongest possible* non-interference lemma that is provable using the underlying abstraction. We then simply check if this invariant implies the safety property. If this approach fails to prove the property, i.e. the computed invariant does not imply the property, then one must necessarily use a better abstraction, for example by adding more concrete processes or axillary variables. In particular since there is no stronger invariant that can be proved invariant using the approach, no manually selected lemmas would have helped out.

Roughly, our approach involves starting with the strong non-interference conjecture *False*. We strengthen the concrete system with this conjecture, abstract, and compute the reachable states in the abstract domain. We then concretize the reachable states, which will yield some weaker conjecture. We iterate, using the successively weaker conjectures found by concretizing the reachable abstract states as lemmas for strengthening the concrete system. Hence our approach involves an inner loop that computes forward reachability in abstract systems, and an outer loop that concretizes the reachable abstract states, strengthens the concrete system with the result, and re-abstracts. We terminate when the outer loop reaches a fix-point in the sense that no new reachable abstract states are found.

Two primary contributions of this paper are a general theory that supports this technique, and an instantiation of the theory for a class of symmetric parameterized protocols, similar to that of Krstić's [15]. This class

The author's email is jesse.d.bingham@intel.com. *This version corrects a minor typo over the submitted version, wherein N was defined to be $m + \ell$, rather than the correct value $m + \ell + 1$.*

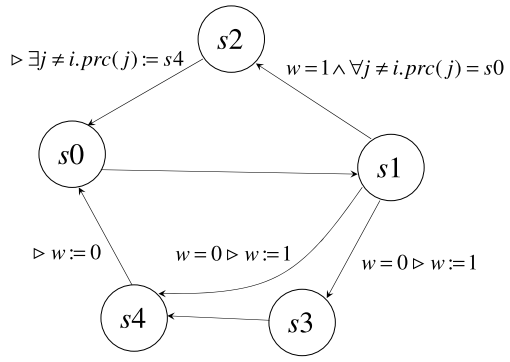

Fig. 1. A process in our example parameterized system

admits a *small model theorem* not unlike that of Pnueli et al. [23], which allows us to perform all necessary computations using BDDs and symbolic model checking [4]. For complex examples, BDD blow up can become an issue. To combat this well-known problem, we propose a variant on our approach wherein only the abstracted transitions (rather than *all* transitions) are strengthened with the non-interference conjectures.

We've implemented an instance of approach using the *forte* [26] formal verification system. The tool takes a protocol described in forte's functional language reFLect, and automatically performs parameterized verification. We have successfully verified properties of three caching protocols: the original GERMAN protocol, a significantly different version GERMAN2004 [19], [13], and the Stanford FLASH protocol [16].

The paper is organized as follows Sect. 2 provides a simple motivating example, which contrasts the classical guard strengthening approach with our approach. Related work is discussed in Sect. 3. The general theory behind our approach is presented in Sect. 4, which again contrasts the classical approach with our own. This theory is instantiated for a class of symmetric parameterized protocols in section 5. Our case studies and further discussion are presented in Sects. 6 and 7, respectively. We note that proofs and some gory technical details have been relegated to appendices.

2 Motivating Example

An example parameterized system is given in Fig. 1. The system is the interleaved composition of an arbitrary number n of identical processes.The system has two state variables: $prc : \mathbb{N}_n \rightarrow \{s0, s1, s2, s3, s4\}$, where $\mathbb{N}_n = \{0, \ldots, n-1\}$ are the process IDs (PID), and w, which is a Boolean. So $prc(i)$ is the "local" state of process i, while w is a "shared" global variable. Initially, all processes are in state $s0$ and w is *false*. The transitions of an arbitrary process i are shown in Fig. 1; each transition is labeled with a guarded command $G \triangleright C$, which means that the transition can occur whenever G is true and C is the resulting update. If G is omitted, then the transition is always enabled; if $\triangleright C$ is omitted, then there is no update other than process i's local state change. Note that some guards and updates mention other processes $j \neq i$. We wish to establish that the following is an invariant:

$$\forall i \neq j. \neg(prc(i) = s4 \wedge prc(j) = s4) \tag{1}$$

The states of the abstract systems are same as the parameterized system, except that there are only two processes 0 and 1. Hence the domain of prc is restricted to the two PIDs $\mathbb{N}_2$. We call these abstract states *views* and write them as a triple $(w, prc(0), prc(1))$. We now outline how both the classic approach and our own approach tackles parameterized verification of this example.

The Classic Approach starts by model checking an initial abstract system, created by conservatively abstracting away all processes except 0 and 1. This yields the following counterexample (i.e. a view sequence)

$$(0,s0,s0), (0,s0,s1), (0,s4,s1), (1,s4,s4)$$

The human realizes that the counterexample stems from an abstracted process transitioning from $s2$ to $s0$, with $j = 0$ in the update. This forces process 0 into $s4$, leading to a violation of (1). By inspection, the human concludes that $s2$ is in fact unreachable, and writes the following lemma:

$$\psi_1 \;=\; \forall i. prc(i) \neq s2 \tag{2}$$

Strengthening the the concrete system with ψ_1 effectively removes the transition from $s2$ to $s0$, which is justified since if $s2$ is unreachable, we don't need any transitions out of it. Abstracting and then model checking, we find that ψ_1 holds, but we get another counterexample of (1):

$$(0,s0,s0), (0,s1,s0), (0,s1,s1),$$
$$(1,s4,s1), (0,s4,s1), (1,s4,s4)$$

Here an abstracted process transitions from $s4$ to $s0$, which "interferes" by clearing w, which allows process 1 to get to $s4$. The user writes another lemma

$$\psi_2 \;=\; \forall i \neq j. w \Rightarrow \neg(prc(i) = s4 \wedge prc(j) = s4)$$

Strengthening with $\psi_1 \wedge \psi_2$ allows us to label the $s4$-$s0$ transition with $\psi_2 \triangleright w := 0$. This disallows an abstracted process from transitioning from $s4$ to $s0$ whenever a concrete process is in $s4$. Model checking for the third time, we find all three of (1), ψ_1, and ψ_2 are invariants. The theory allows us to conclude that (1), as well as ψ_1 and ψ_2, is invariant in the original parameterized system.

Our Approach employs a concretization function which takes a set of views V to a universally quantified formula $\gamma(V)$ that is used to strengthen the concrete system. Intuitively, $\gamma(V)$ characterizes the concrete states such that for all distinct PIDs i and j, the view $(w, prc(i), prc(j))$ is in V. We first model check the abstract system obtained by strengthening the concrete system with the "non-interference conjecture" $\gamma(\emptyset)$, which equals $False$. Strengthening with $\gamma(\emptyset)$ disables all transitions, and hence the set of reachable states is just the singleton of the initial view $R_1 = \{(0,s0,s0)\}$. We now

take $\gamma(R_1)$ as our new, weakened conjecture. Strengthening with $\gamma(R_1)$, abstracting, and computing the reachable states yields $R_2 = R_1 \cup \{(0,s1,s0), (0,s0,s1), (0,s1,s1)\}$. We iterate, now using $\gamma(R_2)$ to strengthen. The reachable views are now

$$R_3 = R_2 \cup \left\{ \begin{array}{l} (1,s3,s0), (1,s4,s0), (1,s3,s1), (1,s4,s1), \\ (1,s0,s3), (1,s0,s4), (1,s1,s3), (1,s1,s4) \end{array} \right\}$$

Strengthening with $\gamma(R_3)$ yields reachable views R_4 where $R_4 = R_3$. Since we've reached a fix-point, our theory tells us that $\gamma(R_3)$ is an invariant of the concrete system. Next we simply check that $\gamma(R_3)$ implies our property, which it does. We note that $\gamma(R_3)$ *properly* implies $\psi_1 \wedge \psi_2$; this is no coincidence as our algorithm computes the strongest lemma provable in the given abstraction/concretization.

3 Related Work

There have been many approaches to parameterized model checking and the more general paradigm of infinite state model checking, for examples well-structured transition systems [11], [12], regular model checking [3], the work of Emerson and Kahlon [9], and that of Pong and Dubois [24]. To our knowledge, none of these have been applied to a system with the complexity of the FLASH protocol. The rest of this section focuses on approaches that are closer to our own.

The idea of using non-interference lemmas for parameterized model checking is attributed to McMillan [20], [21], who added support for this style of reasoning into Cadence SMV. The idea was later used along with Murphi by Chou et al. [6] and formalized further by Krstić [15] and Li [18]. Both papers [21], [6] verify the FLASH coherency protocol [16] using several user-supplied lemmas and the addition of history variables. Similar types of reasoning have been applied by Chen et al. to verify non-parameterized (yet complex) hierarchical protocols [5]. Finally, a simultaneous submission to FMCAD [27] simplifies the process of concocting non-interference lemmas by incorporating the notion of a protocol *flow*. We view this work as complementary to ours, since if our aggressive automatic approach blows-up, then it is desirable to fall back onto an intuitive manual lemma framework.

The most closely related work was done recently by Lv et al. [19]. Like us, they employ BDD-based methods to automatically generate non-interference lemmas. Essentially they use the heuristic for generating candidate "invisible invariants" [23] to generate possible non-interference lemmas. Our work also draws from the invisible invariants work [23] . In particular, our BDD-based concretization computation is inspired by their techniques. Furthermore, we employ a similar small model theorem, though we use slightly tighter syntactic restrictions in order to make the "small" smaller. However, our respective applications of the small model theorems are distinct. We compare against these papers [19], [23] further in Sect. 7.

Other related work includes that of Pandav et al. [22], who have proposed a set of heuristics to aid in constructing invariants for caching protocols. Also, *environment abstraction* [7] uses what amounts to existentially quantified predicate abstraction on the environment of a (concrete) process.

4 Theoretical Foundation

The method of guard strengthening for parameterized verification has been formalized by others [6], [15], [18]. Our re-formalization here is motivated by three factors. First, our formalization is tailored to show how automation is achieved. Second, ours is more general than parameterized systems; it in fact applies to a finite abstraction of any finite or infinite state transition system that is a Galois connection. Finally, we believe this formalization in terms of Galois connections is more succinct than previous explanations.

Given a set S called a *state space*, a *transition system* over S is a triple $\mathcal{T} = (S, I, T)$ where $I \subseteq S$ are called the *initial states* and $T \subseteq S \times S$ is called the *transition relation*. A state $s \in S$ is said to be *reachable* in $\mathcal{T}$ if there exists states $s_0, s_1, \ldots, s_\ell$ such that $s_0 \in I$, $s_\ell = s$, and $(s_i, s_{i+1}) \in T$ for all $0 \leq i < \ell$. The set of all reachable states of $\mathcal{T}$ is denoted $\mathsf{Reach}(\mathcal{T})$. A set $\psi \subseteq S$ is call an *invariant* if $\mathsf{Reach}(\mathcal{T}) \subseteq \psi$. A transition relation $T \subseteq S \times S$ induces a *post image operator* $\mathsf{post}[T] : 2^S \rightarrow 2^S$ defined by $\mathsf{post}[T](\phi) = \{s' \mid \exists s \in \phi.(s, s') \in T\}$. We will identify subsets ϕ of S with logic formulas that characterize them, and call such formulas *state formulas*.

Our goal is to verify that a state formula p is an invariant of $\mathcal{T} = (\mathcal{C}, I, T)$, which we call the *concrete system*. The concrete state space $\mathcal{C}$ consists of the type-consistent assignments to some set of state variables. In the case of parameterized systems, these types depend on a natural parameter n, but this is not relevant for the theoretical development of this section.

One can conjoin a state formula ψ with the concrete transition relation T to strengthen it. Formally, let us define

$$\mathcal{T} \diamond \psi = (\mathcal{C}, I, T \cap (\psi \times \mathcal{C}))$$

We call $\mathcal{T} \diamond \psi$ the *strengthening* of $\mathcal{T}$ by ψ. Intuitively, $\mathcal{T} \diamond \psi$ is the same as $\mathcal{T}$, except that any state $c \in \mathcal{C}$ in which ψ does not hold has no transitions. The following theorem is well-known (see e.g. Krstić [15]) and underlies the approach of using strengthening:

Theorem 1: ψ is invariant in $\mathcal{T}$ if and only if ψ is invariant in $\mathcal{T} \diamond \psi$.

The concrete system is verified by constructing a succession of abstract systems that over-approximate the concrete system. The abstract systems are over a finite set $\mathcal{A}$ called the *abstract domain*, which, along with a partial order $\sqsubseteq$, is a lattice. Each element of $\mathcal{A}$ is an object that represents a certain set of concrete states, and $\sqsubseteq$ corresponds to inclusion of the represented sets. Often $\mathcal{A}$ will be a powerset and $\sqsubseteq$ is $\subseteq$. We assume a *Galois connection* (α, γ) between $2^{\mathcal{C}}$ and $\mathcal{A}$. This means that

$\alpha : 2^{\mathcal{C}} \rightarrow \mathcal{A}$ and $\gamma : \mathcal{A} \rightarrow 2^{\mathcal{C}}$ are monotonic functions such that for all $\phi \subseteq \mathcal{C}$ and $A \in \mathcal{A}$, we have that $\alpha(\phi) \sqsubseteq A$ if and only if $\phi \subseteq \gamma(A)$. We add the additional requirement that $\alpha \circ \gamma$ is the identity.[1] The functions α and γ are respectively referred to as *abstraction* and *concretization*. Intuitively, α defines the mapping that associates elements of the abstract domain with sets of concrete states. We say that a concrete state formula ψ is *representable* if $\psi = \gamma(\alpha(\psi))$. The representable formulas are those that can be abstracted without loosing information.

Since the objects in $\mathcal{A}$ represent sets of concrete states, it is natural to approximate the concrete post image operator by a function over $\mathcal{A}$ called an *abstract post-image*. For $U \subseteq \mathcal{C} \times \mathcal{C}$ there is the notion of a *best* abstract post-image $\mathsf{bap}[U] : \mathcal{A} \rightarrow \mathcal{A}$ defined by

$$\mathsf{bap}[U] \;=\; \alpha \circ \mathsf{post}[U] \circ \gamma \tag{3}$$

Intuitively, $\mathsf{bap}[U]$ first concretizes, then takes the post-image of U in the concrete domain, and then abstracts the result. We say that a function $\mathsf{post} : \mathcal{A} \rightarrow \mathcal{A}$ *abstracts* the concrete transition system $(\mathcal{C}, I, U)$ if for all $A \in \mathcal{A}$ we have that $\mathsf{bap}[U](A) \sqsubseteq \mathsf{post}(A)$. We extend Reach to apply to abstract post-images: if $A_0 \in \mathcal{A}$ and $\mathsf{post} : \mathcal{A} \rightarrow \mathcal{A}$ is a monotonic function, then $\mathsf{Reach}(A_0, \mathsf{post}) = \mathsf{post}^k(A_0)$, where k is minimal such that $\mathsf{post}^{k+1}(A_0) \sqsubseteq \mathsf{post}^k(A_0)$ (note that k must exists since $\mathcal{A}$ is finite). We say that $A \in \mathcal{A}$ is an invariant of (A_0, post) if $\mathsf{Reach}(A_0, \mathsf{post}) \sqsubseteq A$.

Both the classic approach and our own approach employ a mechanism for performing strengthening and abstraction of the concrete transitions. We will denote this mechanism by Ω, which is not made explicit in the classic papers. We do so here since it is a key ingredient in our method, and serves to compare the two approaches. Given a non-interference conjecture ψ, Ω first strengthens the concrete system $\mathcal{T}$ with ψ and then builds an abstract post-image for the strengthened system $\mathcal{T} \diamond \psi$. Formally, Ω takes a representable state formula ψ and returns a function $\Omega(\psi) : \mathcal{A} \rightarrow \mathcal{A}$ such that $\Omega(\psi)$ abstracts $\mathcal{T} \diamond \psi$. We now describe how the classic approach and our approach manifest in this formal setting.

The Classic Approach. Some papers that use the classic approach compute their Ω automatically [21], [19] while others involve the human manually selecting guarded commands to strengthen [6], [27]. Also, the user may strengthen with formulas that are weaker than ψ [6], [15], [27]; this is certainly permissible under our definition of Ω.

Let us say that a representable concrete state formula p is a *provable invariant* if there exists a representable concrete state formula ψ (i.e. the non-interference lemma) such that both $\mathsf{Reach}(\alpha(I), \Omega(\psi)) \sqsubseteq \alpha(\psi)$ and $\mathsf{Reach}(\alpha(I), \Omega(\psi)) \sqsubseteq \alpha(p)$. It is straightforward to show that a provable invariant deserves this title:

1. Classic work on predicate abstraction [14] adds this requirement too.

Theorem 2: If p is a provable invariant, then p is an invariant.

Of course the converse does not hold; the Galois connection or Ω is necessarily too weak to prove many (semantically true) invariants. Theorem 2 succinctly justifies the classical approach, although it hides the fact that the non-interference lemma ψ used to prove p is typically constructed iteratively via counter-example analysis as a conjunction $\bigwedge_i \psi_i$.

Our Approach makes a minor additional monotonicity requirement on Ω. We call Ω *monotonic* if for all representable concrete state formula ψ_1 and ψ_2 such that $\psi_1 \Rightarrow \psi_2$, and for all $A \in \mathcal{A}$, we have that $\Omega(\psi_1)(A) \sqsubseteq \Omega(\psi_2)(A)$. This effectively disallows Ω from giving us a better abstract post-image from a weaker ψ.

Our algorithm can now be described. Let us assume a monotonic Ω, and define the function $\mathsf{Reach}{\circ}\Omega : \mathcal{A} \rightarrow \mathcal{A}$ to be the composition of $\mathsf{Reach}(\alpha(I), \cdot)$ with Ω. I.e. for any $A \in \mathcal{A}$ we define $\mathsf{Reach}{\circ}\Omega(A) = \mathsf{Reach}(\alpha(I), \Omega(\gamma(A)))$ So $\mathsf{Reach}{\circ}\Omega$ maps an element of the abstract domain to the best abstract approximation (afforded by Ω, α, and γ) of the reachable states of the concrete system strengthened with $\gamma(A)$. We note that $\mathsf{Reach}{\circ}\Omega$ is monotonic and acts on a finite domain, hence it has a well defined *least fixed point* (LFP). This LFP is computed as $(\mathsf{Reach} \circ \Omega)^k(\bot)$, where $k \geq 0$ is minimal such that $(\mathsf{Reach} \circ \Omega)^{k+1}(\bot) \sqsubseteq (\mathsf{Reach}{\circ}\Omega)^k(\bot)$ and $\bot$ is the bottom element of $\mathcal{A}$, which is $\emptyset$ if $\mathcal{A}$ is a powerset.

Theorem 3: Assume that Ω is monotonic, let p be a representable concrete state formula, and let Lfp be the least fixed point of $\mathsf{Reach} \circ \Omega$. Then p is a provable invariant if and only if $\mathsf{Lfp} \sqsubseteq \alpha(p)$.

Theorem 3 is the crux of our approach. We simply compute Lfp, and test if $\mathsf{Lfp} \sqsubseteq \alpha(p)$. If so, we can conclude that p is an invariant of the concrete system $\mathcal{T}$. If $\mathsf{Lfp} \sqsubseteq \alpha(p)$ does not hold, then either p is not an invariant, or Ω, α, and/or γ are too weak to prove that p is an invariant.

Automation aside, our approach is in a sense dual to the classical approach. The latter initially doesn't strengthen *at all*, i.e. the strengthened is done with *True*. Then the user repeatedly conjuncts constraints, i.e. strengthens the formula, until a lemma is found that proves the property p. On the other hand, we start with the conjecture *False*, and iteratively weaken it until we have a provable lemma. Since we are left with the strongest lemma (afforded by Ω, α, and γ), we know that no human intervention (respecting Ω) could help.

We can eliminate the possibility that Ω is too weak by using a best Ω relative to (α, γ). Say that Ω is *best* if it always yields the best abstract post image. Formally, this means that for all $A \in \mathcal{A}$, $\mathsf{post}[\Omega(A)] = \mathsf{bap}[T \wedge \gamma(A)]$. If Ω is best, it follows that Lfp is the strongest non-interference lemma that is provable under Galois connection (α, γ). In our instantiation of this theory in the next section, we will first define Ω to be best. However, we have found that our BDD-based methods can potentially blow-up when using a best Ω. In this case we propose

weaker variants that are not best, but are still good enough to automatically prove the desired invariant.

A simple optimization over out algorithm described here is as follows. Rather than computing the LFP of $(\mathsf{Reach} \circ \Omega)$, it is more efficient to compute the LFP of the sequence $A_0, A_1, \ldots$ defined by $A_0 = \alpha(I)$ and $A_{i+1} = \mathsf{Reach}(A_i, \Omega(\gamma(A_i)))$. This means that rather starting each reachable state computation with the initial state abstraction, we start abstract object computed in the previous iteration. This converges on the same fix point Lfp.

5 Parameterized Protocols

The theory of Sect. 4 is general, and applies whenever one is dealing with Galois connections. In this section, we instantiate the theory for a class of symmetric parameterized protocols. We restrict the state variables to be of four basic types, which are the same used by others [6], [15], [27], [23]. The types are Booleans, functions from PIDs to Booleans, PIDs, and functions from PIDs to PIDs; we call the functional state variables *arrays*. The PIDs are $\mathbb{N}_n = \{0, \ldots, n-1\}$ for some arbitrary natural n. Clearly the multiple Booleans can be used to to encode any finite enumerated type. We let $\mathcal{P}(n)$ denote the protocol instance with PIDs being $\mathbb{N}_n$.

Our protocol class is defined by a syntax that enforces symmetry and allows for a certain small model theorem. The syntax is quite similar to that of Krstić [15], and we expect the expressiveness is equivalent. The syntax is expressive enough to admit well-known cache protocol case studies, such as the three protocols we consider in Sect. 6. The class easily allows for so-called *conjunctive guard* and *broadcast* communication primitives, which renders even invariant verification undecidable [10]. Thus using an incomplete verification approach such as abstraction is necessary.

The abstract domain is (a certain subset of) the powerset of a set of objects called *views* [6]. A view is essentially a state of $\mathcal{P}(m)$, where m is a small constant that the user selects, except that PID-valued variables can take on a special value called oth, which represents an unknown PID in $\{m, m+1, \ldots, n\}$. Clearly the set of views, and hence its powerset, are both finite. The Galois connection we employ maps a set of protocol states to its set of constituent views, as well as all permutations of such. Similarly, concretization of a given set of views returns (a formula characterizing) the set of states having its views contained in the given set. We show that there exists a small constant N, such that the views of the strengthened transition relation of $\mathcal{P}(N) \diamond \gamma(V)$ (where V is a set of views) capture the views of the transition relation of $\mathcal{P}(n) \diamond \gamma(V)$ for any $n \geq N$; this is our small model theorem. Here $N = m + \ell + 1$, where ℓ is the number of existentially quantified variables in the transition relation and m is the number of concrete states in the views. The key corollary of the small model theorem is that we can use $\mathcal{P}(N) \diamond \gamma(V)$ to compute the best Ω for our Galois connection. This is done using BDD-operations on $\mathcal{P}(N)$. We note that our small model theorem assumes that $n \geq N$, hence our parameterized model checking in fact establishes that $\mathcal{P}(n)$ is safe for all $n \geq N$, rather than all $n \geq 0$. We regard this as a negligible limitation.

variable set	type in protocol n-state state	type in view
W	$\mathbb{B}$	$\mathbb{B}$
X	$\mathbb{N}_n \to \mathbb{B}$	$\mathbb{N}_m \to \mathbb{B}$
Y	$\mathbb{N}_n$	$\mathbb{N}_m^{oth}$
Z	$\mathbb{N}_n \to \mathbb{N}_n$	$\mathbb{N}_m \to \mathbb{N}_m^{oth}$

TABLE 1
The four types allows for protocol variables, and how they are typed in views.

We now elaborate on these ideas; first describing the concrete syntax in Sect. 5.1, then the abstract domain of sets of views in Sect. 5.2, then the Galois connection in Sect. 5.3, and finally how we compute our Ω in Sect. 5.4. To reduce the flood of notation, we simply denote the Galois connection of this section by (α, γ).

5.1 Concrete System

The parameterized protocol $\mathcal{P}(n)$ has variables from four sets W, X, Y, and Z. We call the variables in $W \cup X \cup Y \cup Z$ the *protocol variables*. The variables are typed according to the middle column of Table 1, where $\mathbb{B}$ are the two Boolean constants. The states of $\mathcal{P}(n)$ are the set of type-consistent assignments to $W \cup X \cup Y \cup Z$. We call these assignments *protocol n-states*. For protocol n-state s and protocol variable ν, we write $s[\nu]$ for the value assigned to ν by s. For $\nu \in X \cup Z$, we write $s[\nu(i)]$ for the ith entry in the array $s[\nu]$.

The transition relation of $\mathcal{P}(n)$, and strengthenings thereof, are expressed as formulas we call *protocol transition formula*. A rigorous definition of protocol formula is given in Appendix A ; here we give a more intuitive description. The important point is that the syntax provides a balance between the two conflicting requirements of expressiveness and having a small enough *small model theorem*, which will be stated in Sect. 5.4. A protocol transition formula is a restricted first order formula over atomic propositions called atoms. These *atoms* perform various queries on the protocol variables, such as indexing into an array, comparison between PID variables, and evaluating a Boolean. Atoms also allow for priming of protocol variables to refer to the next state. We employ three sets of PID variables (disjoint from Y) to quantify over: $E = \{e_0, \ldots, e_{\ell-1}\}$, $U = \{u_0, \ldots, u_{k-1}\}$, and $Q = \{q_0, \ldots, q_{m-1}\}$. We will write $\vec{e}$ for the list $e_0, \ldots, e_{\ell-1}$ and similarly for $\vec{u}$ and $\vec{q}$. Protocol transition formulas are of the form

$$(\exists \vec{e}. \forall \vec{u}. \phi_0) \wedge (\forall \vec{q}. \phi_1) \tag{4}$$

Where ϕ_0 and ϕ_1 are quantifier-free Boolean combinations of atoms with certain constraints (see Appendix A). Here ϕ_0 characterizes the unstrengthened

transition relation; the existential quantification allows for "Murphi"-like rulesets [8] constructs over PIDs. Hence $\ell = |E|$ is the maximum level of nesting of PID rulesets. The second conjunct $\forall \vec{q}.\phi_1$ is the non-interference conjecture. We will see in Sect. 5.3 that our concretization function produces formula of this form; we note that m here is the number of "concrete processes" used in views.

Our initial state formula I will also be required to be protocol transition formula, except no primed variables may appear in I. In our case studies I only uses the second conjunct, since there is no existential quantification necessary.

5.2 Abstract Domain: Symmetric Sets of Views

A *view* is an assignment to the protocol variables, but with typing according to the rightmost column in Table 1. Here $\mathbb{N}_m^{oth} = \{0, \ldots, m-1\} \cup \{oth\}$, where oth is a fresh symbol that will represent an arbitrary element of $\mathbb{N}_n \setminus \mathbb{N}_m$. Our abstract domain is simply the set of *symmetric* sets of views, along with the inclusion ordering $\subseteq$. This notion of symmetric is best defined using the Galois connection, which is the focus of the next subsection. Hence we postpone the definition momentarily.

5.3 Galois Connection

Let τ be a protocol transition formula (4) and let s_0 and s_1 be protocol n-states. We write $(s_0, s_1) \vDash \tau$ to indicate that τ is satisfied when its unprimed and primed free variables are interpreted by s_0 and s_1, respectively. Call an injection $\pi : \mathbb{N}_m \to \mathbb{N}_n$ a *view map*. For any $i \in \mathbb{N}_n$ and view map π, define $\hat{\pi}(i)$ to be $\pi^{-1}(i)$ if i is in the range of π, otherwise $\hat{\pi}(i) = oth$. Now, given view map π, the π-*view* of protocol n-state s is the view $v = \mathsf{View}_\pi(s)$ defined by

- for all $w \in W$ define $v[w] = s[w]$
- for all $x \in X$ and $i \in \mathbb{N}_m$ define $v[x(i)] = s[x(\pi(i))]$
- for all $y \in Y$ define $v[y] = \hat{\pi}(s[y])$
- for all $z \in Z$ and $i \in \mathbb{N}_m$ define $v[z(i)] = \hat{\pi}(s[z(\pi(i))])$

So, essentially, v permutes PIDs so that PIDs in the image of π become the first m PIDs $\mathbb{N}_m$, and PIDs in $\mathbb{N}_n \setminus \mathbb{N}_m$ are replaced with oth.

We now introduce the abstraction function α that takes sets of concrete states to sets of views and a corresponding concretization function γ. For a set of concrete states φ, define

$$\alpha(\varphi) \;=\; \{\mathsf{View}_\pi(s) \mid s \in \varphi \text{ and } \pi \text{ is a view map}\}$$

Thus $\alpha(\varphi)$ is the set of all views that are the π-views of some state in φ for some π. Computing α using BDDs is straightforward, especially when φ is symmetrical. In this case, $\alpha(\varphi)$ can be computed by $\mathsf{View}_{\mathsf{id}}(\varphi)$, where id is the identity function on $\mathbb{N}_m$. In our algorithm, α is only directly applied to such sets, i.e. the initial states and the invariance property.

Concretization is be expressed by

$$\gamma(V) \;=\; \forall \vec{q}.\mathsf{distinct}(\vec{q}) \Rightarrow \bigvee_{v \in V} \Pi(v) \qquad (5)$$

Here $\mathsf{distinct}(\vec{q})$ expresses that $q_i \neq q_j$ for all $0 \leq i < j < m$. For a view v, the formula $\Pi(v)$ is a quantifier-free Boolean combination of atoms that precisely characterizes the concrete states having π-view being v, where π takes $i \in \mathbb{N}_m$ to (a valuation of) each $q_i \in Q$. Intuitively, then, $\gamma(V)$ says that for any m distinct PIDs $\{q_0, \ldots, q_{m-1}\}$ one selects, the corresponding view must be an element of V. Defining $\Pi(v)$ is straightforward but tedious, hence we relegate it to Appendix B. We emphasize that (5) is given for exposition purposes only and is never used directly during computation.

In Sect. 5.2 we defined views and mentioned that our abstract domain is a subset of the powerset of views, i.e. those that are symmetric. We can now succinctly define what this notion means. A set of views V is said to be *symmetric* if $\alpha(\gamma(V)) = V$. We note that our (α, γ) is quite similar to the Galois connection used by Lahiri and Bryant to do universally quantified predicate abstraction [17].

Theorem 4: (α, γ) is a Galois connection between protocol n-states and symmetric sets of views.

5.4 Computing Ω with BDDs

In this section we explain how we the best abstract post-image $\mathsf{bap}[\tau]$ for a protocol transition formula τ. In turn, this allows us to compute the best Ω. In summary, our protocol transition formula admit a *small model theorem*. Recall that ℓ is the number of existentially quantified variables τ (4), and m is the number of "concrete" processes in the views. let $N = m + \ell + 1$, and let us write τ_N for the set of all pairs of protocol N-states (s, s') such that $(s, s') \vDash \tau$.

Theorem 5 (Small Model Theorem): Let τ be a protocol transition formula and let $N = m + \ell + 1$. Then $\mathsf{bap}[\tau] = \mathsf{post}[T]$, where $T = \{(\mathsf{View}_{\mathsf{id}}(s), \mathsf{View}_{\mathsf{id}}(s')) \mid (s, s') \in \tau_N\}$

Theorem 5 allows us to compute a BDD representation of $\mathsf{bap}[\tau]$ as follows. First, we build a BDD for the transition relation τ_N. Next we massage this BDD into a BDD for T by applying $\mathsf{View}_{\mathsf{id}}$ to both the present and next state variables; this involves two steps. First, all X-type and Z-type variables (both primed and unprimed) with indices in $\{m, \ldots, N-1\}$ are existentially abstracted out of τ_N. Second, oth is represented by *any* value $\geq m$, thus we canonicallize so that whenever a PID variable is $\geq m$, we allow it to be any such value; this is accomplished through straightforward BDD operations. The result can easily be seen to represent the T in the statement of Theorem 5. Finally, $\mathsf{post}[T]$ can again be computed using well-known BDD techniques [4].

Applying our approach of Sect. 4 requires us to map an arbitrary symmetric set of views V to a best abstract post-image $\Omega(\gamma(V))$. This post-image must abstract the formula

$$\tau \;=\; (\exists \vec{e}.\forall \vec{u}.\phi_0) \wedge \gamma(V)$$

Note that τ is a valid protocol transition formula (4) since $\gamma(V)$ is of the form $\forall\vec{q}.\phi_1$. Let τ_N and $\gamma_N(V)$ respectively be the BDD for τ and $\gamma(V)$ over protocol N-states. Once we have $\gamma_N(V)$, computing τ_N is straightforward, and thus so too is $\mathsf{bap}[\tau]$ via Theorem 5.

The BDD $\gamma_N(V)$ is computed as follows. Recall that $\gamma(V)$ is defined by (5). Since $\gamma_N(V)$ restricts us to PIDs in $\mathbb{N}_N$, we can replace the universal quantifier and the antecedent $distinct(\vec{q})$ with a conjunction:

$$\gamma_N(V) \;=\; \bigwedge_{\vec{p}} \left(\bigvee_{v \in V} \Pi(v)[\vec{p}/\vec{q}] \right) \tag{6}$$

where $\bigwedge_{\vec{p}}$ ranges over all m-tuples of distinct elements of $\mathbb{N}_N$. We compute BDDs for each $\bigvee_{v\in V} \Pi(v)[\vec{p}/\vec{q}]$ as the post-image $\mathsf{post}[\Theta(\vec{p})](V)$, using a technique due to Pnueli et al. [23], which defines $\Theta(\vec{p})$ as

$$\begin{aligned} \Theta(\vec{p}) \;=\; & \bigwedge_{w \in W} w' \Leftrightarrow w \\ \wedge\; & \bigwedge_{x \in X} \bigwedge_{0 \le i < m} x'(p_i) \Leftrightarrow x(i) \\ \wedge\; & \bigwedge_{y \in Y} \bigwedge_{0 \le i < m} y' = p_i \Leftrightarrow y = i \\ \wedge\; & \bigwedge_{z \in Z} \bigwedge_{0 \le i,j < m} z'(p_i) = p_j \Leftrightarrow z(i) = j \end{aligned} \tag{7}$$

In effect, for each $\vec{p}$ (an m-tuple over $\mathbb{N}_N$), $\Theta(\vec{p})$ constrains the π-view of the primed state to some view in the unprimed state, where $\pi(i) = p_i$ for each $i \in \mathbb{N}_m$. By taking the post-image of V with respect to $\Theta(\vec{p})$, we get the set of all protocol N-states that have π-views in V. Finally, by conjuncting over all $\vec{p}$, we arrive at $\gamma_N(V)$ as required.

We conclude this section by noting that when BDD blow-up becomes an issue, a remedy is to soundly weaken Ω, which can be done in two orthogonal ways. First, one can remove various conjuncts from (7). This amounts to identifying that certain variables are not needed in the non-interference lemma. Second, when the protocol transition formula is a disjunction of guarded commands (e.g. Murphi [8]), one can apply different subsets of the conjuncts in (6) to each command. This allows one to leave unabstracted commands unstrengthened, while applying strengthening to abstracted commands. Previous approaches [19], [15], [6], [27] allow the human to use such heuristics on a lemma-by-lemma basis; in contrast we write a very simple weakening scheme once and our algorithm then proceeds to compute the strongest lemma provable under the weaker Ω. Weakening Ω tends to decrease the number of iterations of the outer loop while increasing the number of iterations of the inner loop of our algorithm.

6 Case Studies

We applied our technique to three coherency protocols: GERMAN, GERMAN2004, and FLASH. The GERMAN protocol was originally poised as a challenge problem for parameterized model checking by German [13], it is rather simple and has been verified many times in the literature [19], [6], [2], [23], [22]. GERMAN2004 is a significant modification of GERMAN and is considerably more complex; it was first formally verified by Lv et al. [19], though they required some manually added history variables. Finally, FLASH [16] is quite well-known. The control property for FLASH was first automatically verified by Lv et al. [19]. As pointed out by Chou et al. [6]: *"FLASH is a good test for any proposed method of parameterized verification; if the method works on FLASH, then there is a good chance that it will also work on many real-world cache coherence protocols."*

example	time	mem	outter	inner
GERMAN	0.3	1.1	19	2,2,...,2
GERMAN2004	15.0	1.6	8	55, 74, 57, 30, 5, 6, 2, 1
FLASH	57.0	3.2	7	75, 51, 43, 39, 36, 9, 1

TABLE 2
Case studies results

We implemented our approach using the *forte* formal verification system [26]. We hand translated the protocols from their Murphi descriptions into *reFLect*, forte's functional language. They were written as protocol transition formula, which required some minor refactoring in some cases[2]. A tarball including our implementation and the case study models is available [1].

We successfully verified the control and data properties for GERMAN, and the control properties for GERMAN2004 and FLASH. For GERMAN, we used a best Ω and BDD-blow-up was a non-issue For the other two protocols, the ability to weaken Ω was very useful to curb BDD blow-up issues, as was dynamic variable reordering [25].

Our results are given in Table 2. All runs were done on a 64-bit linux machine. The columns "time" and "mem" give the runtime in minutes and the memory usage in GB, respectively. The "outer" column gives the number of iterations of the outer loop of our algorithm, i.e. the number of iterations to find the fix-point of Reach_Ω. The "inner" column gives the number of time-steps needed to compute the reachable states in the abstract system, for each iteration of the outer loop. Since the best Ω was used for GERMAN, the inner loop always iterates just twice.

7 Discussion

7.1 Example of Insufficiency

Fig. 2 gives a very simple parameterized protocol for which our approach of Sect. 5 fails, in the sense that the property holds but our algorithm is unable to prove it. Fig. 2 uses the same notation as Sect. 2. Initially, all

2. For example, protocol formula don't allow for a Y variable to be used to index into an array; however by using an existential variable to index into the array, and adding the constraint that the existential equals the Y variable, the same effect can be achieved.

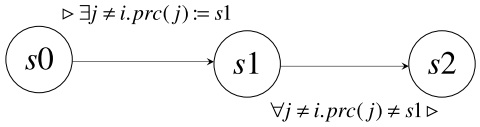

Fig. 2. Example for which our algorithm fails.

processes are in $s0$, and we wish to prove the (clearly true) invariant $\forall i.prc(i) \neq s2$. Our algorithm fails at this verification; this stems from the fact the existentially quantified lemma $\forall i.\exists j \neq i.prc(i) = s1 \Rightarrow prc(j) = s1$ is needed. Our automatically generated lemmas are restricted to be of the form (5), which only allows for universal quantification. We note that adding history variables can help in this situation; if done correctly our algorithm can easily verify this example.

7.2 Comparison with Invisible Invariants

The most salient difference between our approach and that of invisible invariants [23] and Lv et al. [19] is that the lemma we generate is guaranteed to be an invariant, while the other methods heuristically produce a *possible* invariant. This possible invariant is constructed by generalizing the reachable states of a small instance of the concrete system. This leads to the question: are there systems for which invisible invariants fail, while our approach succeeds?

The answer is positive. Consider a parameterized system where each process has a single bit, initially all *False*. There is a global 15-bit counter initially 0. Any process that has its bit clear can set it, and increment the counter (unless it is saturated). We want to prove that if any two processes have their bits set, then the counter is at least 2. The invisible invariants method would compute the reachable states of the concrete system with 2 processes. Generalizing this space does not yield an invariant, However, our algorithm produces an invariant that is strong enough to prove the property. Of course, if a large enough concrete system is used to generate the candidate invisible invariant, their approach will succeed. However, this would require model checking the instance of size 2^{15}, which has over 2^{15} state variables.

REFERENCES

[1] J. Bingham. Appendices and code for this paper. `http://www.cs.ubc.ca/~jbingham/fmcad08.html`.

[2] J. Bingham and A. J. Hu. Empirically efficient verification for a class of infinite-state systems. In *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2005.

[3] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *12th International Conference on Computer Aided Verification (CAV)*, 2000.

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2), 1992.

[5] X. Chen, Y. Yang, G. Gopalakrishnan, and C.-T. Chou. Reducing verification complexity of a multicore coherence protocol using assume/guarantee. In *FMCAD '06: Proceedings of the Formal Methods in Computer Aided Design*, pages 81–88, 2006.

[6] C. Chou, P. K. Mannava, and S. Park. A simple method for parameterized verification of cache coherence protocols. In *5th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, 2004.

[7] E. Clarke, M. Talupur, and H. Veith. Environment abstraction for parmeterized verification. In *7th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI)*, 2006.

[8] D. L. Dill, A. J. Drexler, A. J. Hu, and C. H. Yang. Protocol verification as a hardware design aid. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 522–525, 1992.

[9] E. A. Emerson and V. Kahlon. Exact and efficient verification of parameterized cache coherence protocols. In *Correct Hardware Design and Verification Methods (CHARME)*, 2003.

[10] E. A. Emerson and V. Kahlon. Model checking guarded protocols. In *18th IEEE Symposium on Logic in Computer Science (LICS)*, pages 361–370, June 2003.

[11] J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *14th IEEE Symposium on Logic in Computer Science (LICS)*, pages 352–359, 1999.

[12] A. Finkel and P. Schnoebelen. Well structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.

[13] S. German. Personal correspondence. 2008.

[14] S. Graf and H. Saidi. Construction of abstract state graphs with pvs. In *Proc. 9th International Conference on Computer Aided Verification (CAV'97)*, 1997.

[15] S. Krstić. Parameterized system verification with guard strengthening and parameter abstraction. In *Automated Verification of Infinite-State Systems*, 2005.

[16] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy. The stanford FLASH multiprocessor. In *21st Annual International Symposium on Computer Architecture (ISCA)*, pages 302–313, 1994.

[17] S. K. Lahiri and R. E. Bryant. Constructing quantified invariants via predicate abstraction. In *5th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI)*, pages 267–281, 2004.

[18] Y. Li. Mechanized proofs for the parameter abstraction and guard strengthening principle in parameterized verification of cache coherence protocols. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 1534–1535, 2007.

[19] Y. Lv, H. Lin, and H. Pan. Computing invariants for parameter abstraction. In *MEMOCODE '07: Proceedings of the 5th IEEE/ACM International Conference on Formal Methods and Models for Codesign*, pages 29–38, 2007.

[20] K. L. McMillan. Verification of infinite state systems by compositional model checking. In *Correct Hardware Design and Verification Methods (CHARME)*, pages 219–234, 1999.

[21] K. L. McMillan. Parameterized verification of the FLASH cache coherence protocol by compositional model checking. In *Correct Hardware Design and Verification Methods (CHARME)*, pages 179–195, 2001.

[22] S. Pandav, K. Slind, and G. Gopalakrishnan. Counterexample guided invariant discovery for parameterized cache coherence verification. In *CHARME*, pages 317–331, 2005.

[23] A. Pnueli, S. Ruah, and L. Zuck. Automatic deductive verification with invisible invariants. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 82–97, 2001.

[24] F. Pong and M. Dubois. Formal automatic verification of cache coherence in multiprocessors with relaxed memory models. *IEEE Transactions on Parallel and Distributed Systems*, 11(9):989–1006, September 2000.

[25] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *ICCAD '93: Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, pages 42–47, 1993.

[26] C.-J. H. Seger, R. B. Jones, J. W. O'Leary, T. Melham, M. D. Aagaard, C. Barrett, and D. Syme. An industrially effective environment for formal hardware verification. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 24(9):1381–1405, September 2005.

[27] M. Talupur and M. R. Tuttle. Going with the flow: Parameterized verification using message flows. In *Submitted to FMCAD '08*, 2008.

APPENDIX A
PROTOCOL FORMULA

Definition 1 (atom): Let B and C be sets of variables of type $\mathbb{N}_n$. An (B, C)-*atom* is one of the following formulas

1) w or w' for some $w \in W$
2) $x[p]$ or $x'[p]$, for some $x \in X$ and $p \in B \cup C$
3) $y \doteq c$ or $y' \doteq c$ for some $y \in Y$ and $c \in C$
4) $y' \doteq y$ for some $y \in Y$
5) $b \doteq c$ for some $b \in B$ and $c \in C$
6) $c_1 \doteq c_2$ for some $c_1, c_2 \in C$
7) $z[p] \doteq c$ or $z'[p] \doteq c$ for some $z \in Z$, $c \in C$, and $p \in B \cup C$
8) $z'[b] \doteq z[b]$ for some $z \in Z$ and $b \in B$

Protocol formula are then defined to be those of the form (4), repeated here for convenience:

$$(\exists \vec{e}.\forall \vec{u}.\phi_0) \wedge (\forall \vec{q}.\phi_1)$$

Here $\vec{e}$, $\vec{u}$, and $\vec{q}$ are respectively the lists of variables $e_0, \ldots, e_{\ell-1}$, $u_0, \ldots, u_{k-1}$, and $q_0, \ldots, q_{m-1}$. Intuitively, C are quantified variables that can be used to compare against Y- and Z- type protocol variables and also used to index into X- and Z- type protocol array variables, while the variables of B can only be used for indexing. The sub-formulas ϕ_0 and ϕ_1 are respectively Boolean combinations of (U, E)-atoms and $(\emptyset, Q)$-atoms; further ϕ_1 contains no primed variables. Lastly, we add the constraint that no atom of type 4 or 8 appears under an odd number of negations in ϕ_0.

APPENDIX B
CONCRETIZATION DEFINED

Recall our definition of γ (5):

$$\gamma(A) = \forall \vec{q}.\mathsf{distinct}(\vec{q}) \Rightarrow \bigvee_{v \in V} \Pi(v)$$

Here we formalize the definition of $\Pi(v)$ where v is a view. We define

$$\Pi(v) = \Pi_W(v) \wedge \Pi_X(v) \wedge \Pi_Y(v) \wedge \Pi_Z(v)$$

where

- $\Pi_W(v) = \bigwedge_{w \in W} w^{v[w]}$, where $w^{v[w]}$ is the positive literal w if $v[w] = \mathit{True}$, or the negative literal $\neg w$ otherwise
- $\Pi_X(v) = \bigwedge_{x \in X} \bigwedge_{0 \le i < m} x(q_i)^{v[x](i)}$, where the superscript is interpreted as in the previous bullet
- $\Pi_Y(v) = \bigwedge_{y \in Y} f(y)$, where $f(y)$ is $\bigwedge_{0 \le i < m} \neg q_i \doteq y$ if $v[y] = \mathit{oth}$, otherwise $f(y)$ is $y \doteq q_{v[y]}$
- $\Pi_Z(v) = \bigwedge_{z \in Z} \bigwedge_{0 \le i < m} f(z, i)$ where $f(z, i)$ is $\bigwedge_{0 \le j < m} \neg q_j \doteq z(q_i)$ if $v[z(i)] = \mathit{oth}$, otherwise $f(z, i)$ is $z(q_i) \doteq q_{v[z(i)]}$

APPENDIX C
PROOFS

Theorem 1: ψ is invariant in $\mathcal{T}$ if and only if ψ is invariant in $\mathcal{T} \diamond \psi$.

Proof: A simple induction. □

Theorem 2: If p is a provable invariant, then p is an invariant.

Proof: Let $R = \mathsf{Reach}(\alpha(I), \Omega(\psi))$. There exists a representable concrete state formula ψ such that $R \sqsubseteq \alpha(\psi)$, thus $\gamma(R) \subseteq \gamma(\alpha(\psi)) = \psi$. Similarly we have that $\gamma(R) \subseteq p$. Since $\Omega(\psi)$ abstracts $\mathcal{T} \diamond \psi$, we have that $\alpha(\mathsf{Reach}(\mathcal{T} \diamond \psi)) \sqsubseteq R$ and hence $\gamma(\alpha(\mathsf{Reach}(\mathcal{T} \diamond \psi))) \subseteq \psi$ and $\mathsf{Reach}(\mathcal{T} \diamond \psi) \subseteq \psi$. Similarly we arrive at $\mathsf{Reach}(\mathcal{T} \diamond \psi) \subseteq p$. By Theorem 1, ψ must be an invariant of $\mathcal{T}$, which implies that $\mathsf{Reach}(\mathcal{T} \diamond \psi) = \mathsf{Reach}(\mathcal{T})$. Thus $\mathsf{Reach}(\mathcal{T}) \subseteq p$. □

Theorem 3: Assume that Ω is monotonic, let p be a representable concrete state formula, and let Lfp be the least fixed point of $\mathsf{Reach} \circ \Omega$. Then p is a provable invariant if and only if $\mathsf{Lfp} \sqsubseteq \alpha(p)$.

Proof: (if) Since $\mathsf{Reach}(\alpha(I), \Omega(\gamma(\mathsf{Lfp}))) \sqsubseteq \mathsf{Lfp}$ and $\mathsf{Lfp} = \alpha(\gamma(\mathsf{Lfp}))$, we have that p is a provable invariant with $\psi = \gamma(\mathsf{Lfp})$. (only if) Conversely, suppose p is a provable invariant. Then there exists ψ as in the definition of *provable invariant*. Since $\mathsf{Reach} \circ \Omega(\alpha(\psi)) = \mathsf{Reach}(\alpha(I), \Omega(\gamma(\alpha(\psi)))) = \mathsf{Reach}(\alpha(I), \Omega(\psi)) \sqsubseteq \alpha(\psi)$ (the second equality following from the fact that ψ is representable), we have that $\alpha(\psi)$ is a fix-point of $\mathsf{Reach} \circ \Omega$; similarly we also have that $\mathsf{Reach} \circ \Omega(\alpha(\psi)) \sqsubseteq \alpha(p)$. Using the Knaster-Tarski Theorem, the former implies that $\mathsf{Lfp} \sqsubseteq \alpha(\psi)$. By monotonicity of $\mathsf{Reach} \circ \Omega$, this entails $\mathsf{Lfp} = \mathsf{Reach} \circ \Omega(\mathsf{Lfp}) \sqsubseteq \mathsf{Reach} \circ \Omega(\alpha(\psi)) \sqsubseteq \alpha(p)$. □

Theorem 4: (α, γ) is a Galois connection between protocol n-states and symmetric sets of views.

Proof: Straightforward. □

C.1 Proof of Small Model Theorem 5

This proof requires a few definitions and lemmas. Let Σ_n denote the set of protocol n-states. First there is a notion of a collapsing a "large" protocol n-state $s \in \Sigma_n$ into a protocol N-state $r \in \Sigma_N$, where $n \ge N = m + \ell + 1$.

Definition 2: Let s be a protocol n-state for some $n \ge N$, and let $g : \mathbb{N}_N \to \mathbb{N}_n$ be an injection such that $g(i) = i$ for all $i \in \mathbb{N}_m$. Define $G : \mathbb{N}_n \to \mathbb{N}_N$ so that $G(i) = g^{-1}(i)$ if $i \in g(\mathbb{N}_N)$, otherwise $G(i) = m$. Then the *g-collapse* of s is the state $r \in \Sigma_N$ defined as follows.

- for each $w \in W$, $r[w] = s[w]$
- for each $x \in X$ and $i \in \mathbb{N}_N$, $r[x(i)] = s[x(g(i))]$.
- for each $y \in Y$, $r[y] = G(s[y])$
- for each $z \in Z$, and $i \in \mathbb{N}_N$, $r[z(i)] = G(s[z(g(i))])$

Lemma 1: Let s be an protocol n-state, and let r be its g-collapse for some g as in Def. 2. Then $\mathsf{View}_{\mathsf{id}}(s) = \mathsf{View}_{\mathsf{id}}(r)$.

Proof: Follows from Def. 2. □

Theorem 5 (Small Model Theorem): Let τ be a protocol transition formula and let $N = m + \ell + 1$. Then $\mathsf{bap}[\tau] = \mathsf{post}[T]$, where $T = \{(\mathsf{View}_{\mathsf{id}}(s), \mathsf{View}_{\mathsf{id}}(s')) \mid (s, s') \in \tau_N\}$

Proof: Recalling that $\mathsf{bap}[\tau] = \alpha \circ \mathsf{post}[\tau] \circ \gamma$, we must show that for all symmetric sets of views V that $\alpha(\mathsf{post}[\tau](\gamma(V)) = \mathsf{post}[T](V)$. The $\supseteq$ direction follows from the definitions. For $\subseteq$, let $V_i = \{(\mathsf{View}_{\mathsf{id}}(s), \mathsf{View}_{\mathsf{id}}(s')) \mid (s, s') \vDash \tau \text{ and } s, s' \in \Sigma_i\}$. Thanks to symmetry, clearly we have $T = V_N$; hence we will show that $V_n \subseteq V_N$. Now, τ is of the form (4) and conforms to the definition of Sect. A. We assume without loss of generality that ϕ_0 and ϕ_1 are in CNF. This implies that type 3 and type 6 atoms do not appear negated in either of ϕ_0 and ϕ_1.

Let $(\mathsf{View}_{\mathsf{id}}(s), \mathsf{View}_{\mathsf{id}}(s'))$ be a pair in V_n. Since $(s, s') \vDash \phi$, there exists a tuple $\vec{f}$ over $\mathbb{N}_n$ such that

$$(s, s') \vDash \forall \vec{u}.\forall \vec{q}.(\phi_0 \wedge \phi_1)[\vec{f}/\vec{e}] \tag{8}$$

(here we've legally moved quantifiers around). Let F be the set of naturals appearing in $\vec{f}$, and let $H = F \cup \mathbb{N}_m$ let $g : \mathbb{N}_N \to \mathbb{N}_n$ be the function that takes each $0 \leq i < |H|$ to the ith largest number in H, and each $|H| \leq i < N$ to distinct elements of $\mathbb{N}_n$ so that g is an injection (this can always be done since $N \leq n$). Let us define a corresponding G as in Def. 2.

Now define $r, r' \in \Sigma_N$ to be the g-collapses of s and s', respectively. We claim that $(r, r') \vDash \phi$. Specifically, we claim that

$$(r, r') \vDash \forall \vec{u}.\forall \vec{q}.(\phi_0 \wedge \phi_1)[G(\vec{f})/\vec{e}] \tag{9}$$

where $G(\vec{f})$ is G applied component wise to $\vec{f}$. In order to establish (9), letting $\vec{p}$ and $\vec{t}$ be arbitrary tuples of appropriate length over $\mathbb{N}_N$, we will show

$$(r, r') \vDash (\phi_0 \wedge \phi_1)[G(\vec{f})/\vec{e}, \vec{p}/\vec{u}, \vec{t}/\vec{q}] \tag{10}$$

Using (8) we have that

$$(s, s') \vDash (\phi_0 \wedge \phi_1)[\vec{f}/\vec{e}, g(\vec{p})/\vec{u}, g(\vec{t})/\vec{q}] \tag{11}$$

For a literal θ appearing in the CNF $\phi_0 \wedge \phi_1$, let $\theta_{(10)}$ and $\theta_{(11)}$ respectively be the corresponding literals in the RHS of (10) and the RHS of (11). From (11), all clauses in $\phi_0 \wedge \phi_1$ have a literal θ such that $(s, s') \vDash \theta_{(11)}$. We show that $(r, r') \vDash \theta_{(10)}$, which establishes (10) and thus $(r, r') \vDash \phi$. Our proof case-splits on the form of θ. We note that, depending on which of ϕ_0 or ϕ_1 the literal θ appears in, it may be a (U, E)-atom or a $(\emptyset, Q)$-atom (or the negation of such). We also implicitly use the facts that for all $i \in \mathbb{N}_{|H|}$, $i = G(g(i))$ and for all $h \in H$, $h = g(G(h))$. Note that because of the restrictions on type 3 and 6 atoms from Sect. A, the corresponding cases here don't need to handle the negated literal.

We start by considering the case of θ being a (U, E)-atom literal, and case split according to the eight atom flavors given in Def. 1 (except that we typically prove for positive and negative literals for unprimed variables, and simply mention that the primed variants are handled analogously):

1) w or $\neg w$ for some $w \in W$. Trivial, since $s[w] = r[w]$.
2) $x[a]$ or $\neg x[p]$, for some $x \in X$ and $a \in U \cup E$. If $a = e_i$, then $\theta_{(10)}$ is $x(G(f_i))$ and $\theta_{(11)}$ is $x(f_i)$. Since $f_i \in H$, we have $r[x(G(f_i))] = s[x(g(G(f_i)))] = s[x(f_i)]$, the result follows. On the other hand, if $a = u_i$, then $\theta_{(10)}$ is $x(p_i)$ and $\theta_{(11)}$ is $x(g(p_i))$. Since $r[x(p_i)] = s[x(g(p_i))]$, we are done.
3) $y \doteq e_i$ or $\neg y \doteq e_i$ for some $y \in Y$ and $e_i \in E$. If θ is a positive literal, then $\theta_{(10)}$ is $y \doteq G(f_i)$ and $\theta_{(11)}$ is $y \doteq f_i$. Thus $s[y] = f_i$, and since r is defined so that $r[y] = G(s[y])$, the result follows. Now if θ is negative, then we note that $s[y] \neq f_i$ implies that $r[y] = G(s[y]) \neq G(f_i)$, and the result follows.
4) $y' \doteq y$ for some $y \in Y$. Since $s[y'] = s[y]$ implies that $r[y'] = r[y]$, the result follows.
5) $u_j \doteq e_i$ or $\neg u_j \doteq e_i$ for some $u_j \in U$ and $e_i \in E$. Then $\theta_{(10)}$ is $p_j \doteq G(f_i)$ and $\theta_{(11)}$ is $g(p_j) \doteq f_i$. Since $g(p_j) = f_i$ iff $p_j = G(f_i)$, the result follows for either polarity of literal.
6) $e_j \doteq e_i$ or $\neg e_j \doteq e_i$ or for some $e_j, e_i \in E$. Then $\theta_{(10)}$ is $G(f_j) \doteq G(f_i)$ and $\theta_{(11)}$ is $f_j \doteq f_i$. Since $f_j = f_i$ iff $G(f_j) = G(f_i)$, the result follows for either polarity of literal.
7) $z[u_j] \doteq e_i$ for some $z \in Z$, $e_i \in E$, and $u_j \in U$. Then $\theta_{(10)}$ is $z[p_j] \doteq G(f_i)$ and $\theta_{(11)}$ is $z[g(p_j)] \doteq f_i$. Since $s[z(g(p_j))] = f_i$ iff $r[z(p_j)] = G(f_i)$, the result follows. For the case of θ being $z[e_j] \doteq e_i$ for some $z \in Z$, $e_j, e_i \in E$, $\theta_{(10)}$ is $z[G(f_j)] \doteq G(f_i)$ and $\theta_{(11)}$ is $z[f_j] \doteq f_i$. Since $s[z(f_j)] = f_i$ iff $r[z(G(f_j))] = G(f_i)$, the result follows.
8) $z'[u_i] \doteq z[u_i]$ for some $z \in Z$ and $u_i \in U$. Then $\theta_{(10)}$ is $z'[p_i] \doteq z[p_i]$ and $\theta_{(11)}$ is $z'[g(p_i)] \doteq z[g(p_i)]$. Since $s[z'(g(p_i))] = s[z(g(p_i))]$ implies that $r[z'(p_i)] = r[z(p_i)]$, the result follows.

Analogous arguments handle the cases involving primed variables; analogous (but simpler) arguments handle cases where θ is a $(\emptyset, Q)$-atom (i.e. a literal from ϕ_1).

□