Parameterized Verification of Deadlock Freedom in Symmetric Cache Coherence Protocols

Brad Bingham and Mark Greenstreet
Department of Computer Science, University of British Columia
201-2366 Main Mall, Vancouver, B.C., Canada, V6T 1Z4
{binghamb, mrg}@cs.ubc.ca

Jesse Bingham
Intel, Oregon, U.S.A.
jesse.d.bingham@intel.com

***Abstract*—An important problem in the verification of hardware protocols is that of proving deadlock freedom. We view deadlock freedom as the property that for all reachable states, there exists some path to a quiescent state, i.e. one wherein all resources of interest are free and thus all prior requests have been resolved. We establish a framework for showing this property in a class of symmetric parameterized systems. Our approach is based on a mixed abstraction system than includes both an over-approximate and an under-approximate transition relation. Model checking is employed to compute all states reachable through overapproximate transitions, and from each of these states finds a path of underapproximate transitions to a quiescent state. When this fails because the under-approximation is too strong, we provide techniques to suggest additional transitions that can be introduced to soundly weaken the under-approximation. This approach can be viewed as an extension of the well-known approach of guard strengthening for verifying state invariants of parameterized systems. We present proof of deadlock freedom of the German and FLASH cache-coherence protocols as case studies using a semi-automated heuristic tool that mitigates the human effort.**

I. Introduction

Designing distributed protocols is known to be among the trickiest aspects of modern hardware design. A well-known problem that can arise in such systems is *deadlock*, which occurs when a state is reached that involves an unbreakable cyclic dependency between resources [1]. Here a *resource* might be an entry in a transaction table, a slot in a network, etc. We assume that one can easily characterize the *quiescent* states by a state predicate Q; the quiescent states are those wherein all resources are free and there are no in-flight transactions. A deadlock state, then, is simply a state from which there is no path to a quiescent state. Hence we can express deadlock freedom in Computation Tree Logic (CTL) by $\mathsf{AG}\,\mathsf{EF}\,Q$ (i.e. for all reachable states there exists a path to a Q-state).

Model checking [2] is a popular method for verifying that a system adheres to some specification. Classical model checking assumes that the system under consideration is finite-state. However, many researchers have explored techniques to generalize model checking to verify various classes of *parameterized* systems. For the purposes of this paper, a parameterized system $\mathcal{P}$ is a function that yields a finite-state system $\mathcal{P}(n)$ for all naturals $n \geq 1$. Here n indicates the number of values involved in some type P (called the *parametric type*) used by the system, for examples client IDs or addresses. A parameterized model checking problem asks if $\mathcal{P}(n)$ satisfies some given specification for all n[1]. Unless one puts severe restrictions on the class of systems, parameterized model checking is undecidable [3].

A promising approach to parameterized model checking is based on abstraction and compositional reasoning [4], [5], [6], [7], [8], [9], [10], [11] and is typically used to verify universally-quantified (over P) state assertions roughly as follows. An initial abstraction $\mathcal{A}_0$ is created from the syntax of $\mathcal{P}$. By construction, the transitions of $\mathcal{A}_0$ over-approximate those of $\mathcal{P}(n)$ for arbitrary $n > k$ (for some typically small k). If the state assertion holds of $\mathcal{A}_0$, then we can soundly conclude it holds too for $\mathcal{P}(n)$. However, often we are not so lucky and we must *strengthen* the transitions of $\mathcal{P}_0$ using a conjectured state invariant φ_1, yielding a tighter abstraction $\mathcal{A}_1$. This process iterates until we obtain a $\mathcal{A}_j$ wherein the original state assertion along with all of $\varphi_1, \ldots, \varphi_j$ hold. At this point our parametric verification goal has been achieved.

Our work extends this approach to handle deadlock freedom, i.e. properties of the form $\mathsf{AG}\,\mathsf{EF}\,Q$ for a state predicate Q. The key idea is to not only construct abstract transition relations that over-approximate those of $\mathcal{P}(n)$, but also transition relations that *under-approximate* $\mathcal{P}(n)$. The resulting verification framework is formalized in terms of *mixed-abstractions* [12] – systems with two transition relations O and U, which are respectively over-approximative and under-approximative. As in the traditional approaches, O is used to explore the reachable abstract states, which represent an over-approximation of the reachable states of $\mathcal{P}(n)$. However, during this exploration, we explore paths of U to check that the existential CTL formula $\mathsf{EF}\,Q$ holds for each reachable abstract state. If so, it is safe to conclude deadlock freedom of $\mathcal{P}(n)$; because the existence of a path in U implies that of a corresponding path in $\mathcal{P}(n)$. Initially, U can be constructed by checking syntactic properties of transitions of O. Then, analogous to how O might be too *weak*, the initial U might be too *strong*. A key contribution we present are a set of heuristic methods that allow the user to soundly weaken U in these cases. Another contribution is a theorem

[1]Many approaches, including ours, only verify $\mathcal{P}(n)$ for all $n \geq n_0$, where n_0 is a small constant. This is not a shortcoming since the $\mathcal{P}(n)$ where $n < n_0$ are either "uninteresting" or can be dispatched by finite-state model checking.

that supports deadlock freedom verification for Q involving universal quantification over the parametric type.

Ideally, one would seek to establish Linear Time Logic (LTL) *response* properties [13] of the form $\mathsf{G}\,(req \rightarrow \mathsf{F}\,resp)$, where req and $resp$ respectively mean a some request is sent and the corresponding response is received. However, response properties are difficult to model check for parameterized systems and require both computationally and conceptually complex approaches (we review some in Sect. II). We see our deadlock freedom verification as a lighter-weight alternative that is also practically relevant. Indeed, experience working with real hardware protocols in industry indicates that response failures are almost always caused by deadlocks (i.e. violations of $\mathsf{AG}\,\mathsf{EF}\,Q$) rather than more subtle "live-lock" style failures.

II. Related Work

There have been many previous efforts to extend compositional techniques to parameterized safety property verification [4], [5], [6], [7], [8], [9], [10], [11]. As for liveness-like properties, there are several notable works.

McMillan's work on using compositional methods for LTL liveness properties [14] was applied to parametric liveness verification of the FLASH coherence protocol [5]. Although this paper focuses on a proof of safety, the same framework was used to show that whenever the directory is in the pending state, it is eventually not pending [15]. This proof relies on a handful of lemmas and fairness assumptions, designed and proven within SMV.

Fang *et al.* proposed an interesting technique called *invisible ranking* [16] which attempts to automatically guess ranking functions to prove response properties. The associated proof obligations (from [13]) are decided using some small-model theorems and BDDs. The authors have previously used counter abstraction for parameterized liveness verification [17].

Baukus *et al.* employ *WS1S* (a decidable second-order logic) to perform liveness verification of parameterized systems [18], and verify response properties for the German protocol as a case study [19]. Like our approach, human effort is required, to select both abstract predicates and ranking predicates needed to create an appropriate abstraction. The complexity of deciding WS1S is well-known to be super-exponential, hence scalability of this approach seems unlikely.

The earliest example we could find where both over-approximative and under-approximative abstractions of a transition system are employed for verification is the work of Larsen and Thomsen [20]. They distinguish between *necessary* and *admissible* transitions; for a process to refine another it must over-approximate the former and under-approximate the latter. Of course our interests are in *abstraction* rather than *refinement*, which are in a sense inverses of each other. Later the work of Dams *et al.* [12] and independently Cleaveland *et al.* [21] used *mixed transition systems* which are defined with *two* transition relations to formulate abstractions that preserve both universal and existential properties of the modal μ-calculus; our mixed-abstractions are very similar.

III. Preliminaries

This section presents the formal framework that we use to verify quiescence properties of parameterized systems. Section III-A introduces *mixed abstractions* that have two transition relations: one that under-approximates the behaviors of the concrete systems and another that provides an over-approximation. Section III-B presents the idea of *insufficiency* – a mixed-abstraction may have an under-approximation that is too strong to verify the desired quiescence property or an over-approximation that is too weak. Section III-C describes parameterized systems.

A. Systems and Mixed Abstractions

A *system* $\mathcal{S}$ is a tuple (S, I, T) where S is a set of *states*, $I \subseteq S$ is the set of the initial states, and $T \subseteq S \times S$ is the *transition relation*. We write $s_1 \rightsquigarrow_T s_2$ to denote that $(s_1, s_2) \in T^*$. A state s is said to be $\mathcal{S}$*-reachable* (or simply *reachable* if $\mathcal{S}$ is understood) if $s_0 \rightsquigarrow_T s$ for some $s_0 \in I$. A *state predicate* p is simply a subset of S; if $s \in p$ we call s a p*-state*. Following standard CTL syntax, for state predicates p and q, we write: $\mathcal{S} \models \mathsf{AG}\,p$ if all reachable states are p-states; $\mathsf{AG}\,(p \rightarrow \mathsf{EF}\,q)$, if for all reachable p-states s there exists a q-state s' such that $s \rightsquigarrow_T s'$; and $\mathsf{AG}\,\mathsf{EF}\,q$ to mean $\mathsf{AG}\,(true \rightarrow \mathsf{EF}\,q)$.

To show that $\mathsf{AG}\,(p \rightarrow \mathsf{EF}\,q)$ can be inferred for a concrete system, $\mathcal{S}$, by establishing properties of an abstraction, $\mathcal{A}$, we employ Lynch and Vaandrager's notion of *forward simulation* [22]. Let $\mathcal{S}_1 = (S_1, I_1, T_1)$ and $\mathcal{S}_2 = (S_2, I_2, T_2)$ be two systems and $\theta \in S_1 \times S_2$ be an abstraction relation. We say that T_2 forward simulates (or "simulates" for short) T_1 if for every $(s_1, s_1') \in T_1$ and for all s_2 such that $(s_1, s_2) \in \theta$, there is a $s_2' \in S_2$ such that $s_2 \rightsquigarrow_{T_2} s_2'$ and $(s_1', s_2') \in \theta$. This allows system $\mathcal{S}_2$ to take multiple steps that may be invisible in $\mathcal{S}_1$ including possibly steps that have no "explanation" in $\mathcal{S}_1$. This general sense of simulation is motivated by our goal of showing the existence of trajectories. Now let $s_1, \ldots, s_\ell$ be a T_1-path, and suppose T_2 simulates T_1 with respect to θ. By induction on ℓ, there exists an T_2-path $s_1', \ldots, s_k'$ and a non-decreasing surjection $f : \{1, \ldots, \ell\} \rightarrow \{1, \ldots, k\}$ such that $(s_i, s'_{f(i)}) \in \theta$ for all $1 \leq i \leq \ell$. In this case we say that $s_1', \ldots, s_k'$ is a θ*-simulation* of $s_1, \ldots, s_\ell$.

To show $\mathsf{AG}\,(p \rightarrow \mathsf{EF}\,q)$ using abstraction, the abstract system must, for soundness, *over-approximate* the set of reachable p-states, and under-approximate the set of paths from p-states to q-states. Thus we introduce a *mixed abstraction* as defined below.

Definition 1: Let $\mathcal{S} = (S, I, T)$ be a system and let $Reach$ be the $\mathcal{S}$-reachable states. A *mixed abstraction* of $\mathcal{S}$ (relative to $\theta : S \rightarrow S_\mathcal{A}$) is a quadruple $\mathcal{A} = (S_\mathcal{A}, I_\mathcal{A}, U, O)$ such that

- $S_\mathcal{A}$ is a set of abstract states,
- $I_\mathcal{A} \subseteq S_\mathcal{A}$ are the initial abstract states and satisfy $\theta(I) \subseteq I_\mathcal{A}$,
- $O \subseteq S_\mathcal{A} \times S_\mathcal{A}$ simulates T with respect to $\theta \cap (Reach \times S_\mathcal{A})$, and
- T simulates $U \subseteq S_\mathcal{A} \times S_\mathcal{A}$ with respect to $(\theta \cap (Reach \times S_\mathcal{A}))^{-1}$.

Note that we require θ to be a function. Here U and O are respectively called the *under-approximative* (UA) and *over-approximative* (OA) transition relations of the mixed abstraction. When discussing mixed-abstractions, we will often refer to $\mathcal{S} = (S, I, T)$ as the *concrete system*, to S as the *concrete states*, etc.

The following serves as the basis for our approach to proving deadlock freedom for a class of parameterized systems. Let p and q be state predicates over $S_{\mathcal{A}}$, and suppose for all $(S_{\mathcal{A}}, I_{\mathcal{A}}, O)$-reachable p-states s there exists a q-state r such that $s \rightsquigarrow_U r$. We assert this by writing

$$\mathcal{A} \models \mathsf{AG}\,(p \rightarrow \mathsf{EF}\,q) \tag{1}$$

Hence, (1) holds of a mixed abstraction if for all p-states reachable using the over-approximative transition relation, there exists a path through the under-approximative transition relation to a q state.

Lemma 1: Let $\mathcal{A}$ be a mixed abstraction of $\mathcal{S}$ relative to θ and let p and q be state predicates on $S_{\mathcal{A}}$. If $\mathcal{A} \models \mathsf{AG}\,(p \rightarrow \mathsf{EF}\,q)$ then $\mathcal{S} \models \mathsf{AG}\,(\theta^{-1}(p) \rightarrow \mathsf{EF}\,\theta^{-1}(q))$.

Proof: Let $Reach$ be the set of $\mathcal{S}$-reachable states. Let w be any $\theta^{-1}(p)$-state in $Reach$. Because O simulates T with respect to $\theta \cap (Reach \times S_{\mathcal{A}})$, $\theta(w)$ is $(S_{\mathcal{A}}, I_{\mathcal{A}}, O)$-reachable, and furthermore $\theta(w)$ is clearly a p-state. Let $a_0, \ldots, a_m$ be a U-path from $\theta(w) = a_0$ to a q-state a_m. Because T simulates U with respect to $(\theta \cap (Reach \times S_{\mathcal{A}}))^{-1}$, for all $0 \leq i < m$ and all $w_i \in \theta^{-1}(a_i)$ there exists $w_{i+1} \in \theta^{-1}(a_{i+1})$ such that $w_i \rightsquigarrow_T w_{i+1}$. Therefore, taking $w_0 = w$, there is a path $w \rightsquigarrow_T w_m$ where $w_m \in \theta^{-1}(q)$. ∎

Note that the definition of mixed-abstraction explicitly mentions the reachable states of $\mathcal{S}$ (in the involved simulation relations), this is just our means of formalizing the minimal requirements a mixed-abstraction must satisfy in order to prove Lemma 1. In other words, any methodology that aims to construct mixed-abstractions must guarantee *at least* these simulations. We emphasize that this is different than asking the user of such a methodology to precisely characterize $Reach$ (indeed our methodology does not make such a demand).

When performing reasoning that allows us to add (or remove) transitions from U and O in a mixed abstraction, we often will employ the following sufficient conditions. We conclude this section by stating a connection between $\mathcal{S}$ and the transitions of U and O.

Lemma 2: Suppose $\mathcal{S} = (S, I, T)$, $S_{\mathcal{A}}$, $I_{\mathcal{A}}$, and $\theta : S \rightarrow S_{\mathcal{A}}$ are as in Def. 1. If $U, O \subseteq \mathcal{A} \times \mathcal{A}$ satisfy

1) for all $(w, w') \in T$ such that w is $\mathcal{S}$-reachable we have $(\theta(w), \theta(w')) \in O$, and
2) $(s, s') \in U$ implies for all $\mathcal{S}$-reachable $w \in \theta^{-1}(s)$ there exists $w' \in \theta^{-1}(s')$ such that $w \rightsquigarrow_T w'$.

then $(S_{\mathcal{A}}, I_{\mathcal{A}}, O, U)$ is a mixed-abstraction for $\mathcal{S}$.

Proof: Follows trivially from Def. 1.

B. Insufficiency

Lemma 1 allows us to infer $\mathcal{S} \models \mathsf{AG}\,(\theta^{-1}(p) \rightarrow \mathsf{EF}\,\theta^{-1}(q))$ if model checking (or any other means) verifies $\mathcal{A} \models \mathsf{AG}\,(p \rightarrow \mathsf{EF}\,q)$. However, the converse of the lemma does not hold in general (or even in common cases). Let us call the mixed-abstraction $\mathcal{A}$ *insufficient* if $\mathcal{S} \models \mathsf{AG}\,(\theta^{-1}(p) \rightarrow \mathsf{EF}\,\theta^{-1}(q))$ holds but $\mathcal{A} \not\models \mathsf{AG}\,(p \rightarrow \mathsf{EF}\,q)$. If $\mathcal{A}$ is insufficient, it follows that there exists a p-state a_p such that $a_0 \rightsquigarrow_O a_p$ for some $a_0 \in I_{\mathcal{A}}$ but there is no q-state a_q such that $a_p \rightsquigarrow_U a_q$. There are two common causes for insufficiency:

- **OA insufficiency.** There is no (S, I, T)-reachable $s_p \in S$ such that $(s_p, a_p) \in \theta$. Hence a_p does not abstract any reachable state of $\mathcal{S}$. This is often caused by O being too *weak*, i.e. there exists a proper subset $O' \subset O$ such that $(S_{\mathcal{A}}, I_{\mathcal{A}}, O', U)$ is a mixed abstraction of $\mathcal{S}$ wherein a_p becomes unreachable.
- **UA insufficiency.** There *is* (S, I, T)-reachable $s \in S$ such that $(s, a) \in \theta$, however none of the T-paths $s = s_0, \ldots, s_\ell$ where $s_\ell \in \theta^{-1}(q)$ (at least one such T-path must exist), are simulations of any U-paths. This is often caused by U being too *strong*, i.e. there exists a proper superset $U' \supset U$ such that $(S_{\mathcal{A}}, I_{\mathcal{A}}, O, U')$ is a mixed abstraction of $\mathcal{S}$. Here the transitions introduced in U' would be sufficient to ensure the existence of a U'-path that $s_0, \ldots, s_\ell$ is a simulation of.

For the mixed abstractions we use to verify our parameterized systems, we will observe that OA insufficiency is solved in the previous literature, however UA insufficiency is not. Our basic approach is to identify UA insufficiency from a counter-example trace. In practice, the transitions from O that are needed in U are apparent from this counter-example. The basic idea is to show that for each such transition of O, there is a corresponding *path* in the concrete system. Sections IV and V present how this can be done by syntactic pattern matching and model checking of the *abstract* system for properties with the form shown by formula (1).

There is also a third flavor of insufficiency,

- **Abstract quiescence insufficiency.** Note that in Lemma 1, the quiescent predicate actually verified is of the form $\theta^{-1}(q)$, where q is a predicate on the abstract states. Suppose, however that there does not exist a q such that $\theta^{-1}(q)$ characterizes the desired set of quiescent concrete states. We experience this for our case studies; the desired quiescence predicate involves a universal quantification over the parametric type that the underlying simulation relation cannot precisely characterize. That is, if the concrete quiescent states are characterized by a predicate of the form $\forall i.\ \phi(i)$, then there is no abstract predicate q such that $\theta^{-1}(q) = \forall i.\ \phi(i)$. We deal with this form of insufficiency via Theorem 1.

C. Parameterized Systems

For the purposes of this paper, a parameterized system $\mathcal{P}$ is a function mapping natural numbers to systems. We write $\mathcal{P}(n) = (S(n), I(n), T(n))$ to denote the components of $\mathcal{P}(n)$ for an arbitrary n. The states $S(n)$ are the type-consistent assignments to a set of state variables. For a state w of a parameterized system and a state variable v, we write $w.v$ to denote the value w assigns to v. We allow four types of variables:

- finite types that are independent of n, such as booleans and enumerations; for simplicity, we denote all such types as B
- a type which has cardinality n, denoted P_n
- arrays indexed by P_n with elements in B, denoted $\mathsf{array}\,[\mathsf{P}_n]\ \mathsf{of}\ \mathsf{B}$
- arrays indexed by P_n with elements in P_n, denoted $\mathsf{array}\,[\mathsf{P}_n]\ \mathsf{of}\ \mathsf{P}_n$

In this paper we identify the set P_n with the numbers $\{1, \ldots, n\}$ called *nodes*; the only operations supported on nodes are equality comparison, assignment, and nondeterministic choice.[2] This ensures that for all n, $\mathcal{P}(n)$ is fully symmetric [23] in P_n; here we give a brief review of this notion. Let us write $\lambda i.e$ to denote the array a indexed by P_n where $a[i] = e$ and e is an expression of the appropriate type. Let π be a permutation on P_n. We overload π to act on $w \in S(n)$ by defining $\pi(w) \in S(n)$ to be the state such that for each state variable v, $\pi(w).v$ is equal to:

- $w.v$, if v has type B
- $\pi(w.v)$, if v has type P_n
- $\lambda i.(w.v)[\pi^{-1}(i)]$, if v has type $\mathsf{array}\,[\mathsf{P}_n]\ \mathsf{of}\ \mathsf{B}$
- $\lambda i.\pi((w.v)[\pi^{-1}(i)])$, if v has type $\mathsf{array}\,[\mathsf{P}_n]\ \mathsf{of}\ \mathsf{P}_n$

Then $(S(n), I(n), T(n))$ is called *fully symmetric* if for all $w, w' \in S(n)$ and all permutations π on P_n we have both that $w \in I(n)$ iff $\pi(w) \in I(n)$, and $(w, w') \in T(n)$ iff $(\pi(w), \pi(w')) \in T(n)$. The following lemma has a simple inductive proof using the latter.

Lemma 3 (Path Symmetry): For $w, w' \in S(n)$ we have $s \rightsquigarrow_{T(n)} w'$ if and only if $\pi(w) \rightsquigarrow_{T(n)} \pi(w')$.

In Section IV-A, we impose restrictions on parameterized systems in order to be admissible for our method.

IV. Syntactical Abstraction

We assume that the parameterized system is modeled by Murφ [24] or a similar guarded-command notation. Given a program P to describe the parameterized system, we use well-established techniques [4], [5], [6], [7], [8], [9], [10], [11] to obtain an abstraction of P. Our formulation is inspired by the Krstic's "syntactic" approach [7]; Section IV-A states restrictions that we assume on the form of P, and Section IV-B summarizes the abstraction technique. In Section V, we show how the abstraction can be generalized to produce an under-approximate transition relation, U, and how U can be soundly weakened to prove quiescence properties.

A. Syntax and Restrictions

We assume that the guarded-command program that models the parameterized system satisfies certain syntactic restrictions described in this section. These restrictions ease the syntactical abstraction process and simplify reasoning about the program because many useful properties are guaranteed by construction. From the case studies reported in Section VI, we've found that these restrictions are not problematic in practice.

[2]If i and j are nodes, a parameterized system is not allowed to perform a comparison like $i < j$ or perform an incrementation $i := i + 1$.

We say that such a program is *admissible*, and we write AP as a shorthand for an admissible program. An AP has set of variables of the types indicated in Table I. A state of the AP is a type-consistent assignment of values to these variables. If e is a term, we write $s(e)$ to denote the value of e in state s. In Murφ, a guarded command is called a *rule* and has the form: *guard* $\twoheadrightarrow$ *action*, where the *guard* is a boolean-valued expression, and the *action* is a sequence of one or more assignments. We write $r : \rho \twoheadrightarrow \boldsymbol{a}$ to denote rule r with guard ρ and action $\boldsymbol{a}$.

The denotation $[\![r]\!]$ of r is the set of tuples $(s, s') \in S^2$ such that $s(\rho)$, and s' is the state reached by performing action $\boldsymbol{a}$ from state s. Murφ has *rulesets* of the form:

```
ruleset i in P_n do r(i) end;
```

where $r(i)$ is a rule (or a ruleset, as they may be nested). Here, $i \in \mathsf{P}_n$ is called the *ruleset parameter*. If rs is the ruleset indicated above, then

$$[\![rs]\!] \quad = \quad \{(s, s') \mid \exists i \in \mathsf{P}_n.\ (s, s') \in [\![r(i)]\!]\} \qquad (2)$$

A *local boolean predicate* L is a propositional formula over the variables of type $\mathsf{array}\,[\mathsf{P}_n]\ \mathsf{of}\ \mathsf{B}$. For node i, we say $L[i]$ holds of a state if L evaluates to true when its variables are assigned according to the i^{th} array entries of the state. An *admissible program* (AP) must satisfy the following syntactic restrictions. Rulesets have guards that are a conjunct of:

- Boolean terms, composed of variables of type B or $\mathsf{array}\,[\mathsf{P}_n]\ \mathsf{of}\ \mathsf{B}$ indexed by a ruleset parameter, and the logical connectives `AND`, `OR` and `NOT`.
- At most one *forall condition*, of the form $\forall i \in \mathsf{P}_n.\ C[i]$ where C is a local boolean predicate.
- Any number of P-comparisons, of the form $v_1 = v_2$ or $v_1 \neq v_2$, where v_1 and v_2 are variables of type P or $\mathsf{array}\,[\mathsf{P}]\ \mathsf{of}\ \mathsf{P}$ indexed by a ruleset parameter, or a ruleset parameter. Without loss of generality, we restrict each ruleset parameter to appear in at most one P-comparison of equality.

The initial states and ruleset commands given by a sequence assignments of the following forms:

- Assignments of the form $b_1 := b_2$, $a^1_{\mathsf{B}}[i] := b_2$, $a^1_{\mathsf{B}}[i] := a^2_{\mathsf{B}}[i]$, where b_1 and b_2 are variables of type B, a^1_{B} and a^2_{B} are variables of type $\mathsf{array}\,[\mathsf{P}_n]\ \mathsf{of}\ \mathsf{B}$, and i is a ruleset parameter. RHS values may also be the constants true and false.
- Assignments of the form $p_1 := p_2$, $a^1_{\mathsf{P}}[i] := p_2$, $a^1_{\mathsf{P}}[i] := a^2_{\mathsf{P}}[i]$, where p_1 and p_2 are variables of type P_n, a^1_{P} and a^2_{P} are variables of type $\mathsf{array}\,[\mathsf{P}_n]\ \mathsf{of}\ \mathsf{P}_n$, and i is a ruleset parameter.
- *Forall updates* of the form $\forall i \in \mathsf{P}_n.\ a_{\mathsf{B}}[i] := \ell(i)$, where ℓ is a boolean function depending on variables of type B, P_n and on the i^{th} index of array variables.

A BNF grammar for this restriction of Murφ is given in the Appendix.

These restrictions ensure that guards in APs do not contain disjunctions of comparisons between variables of type P and have no existentially quantified terms; updates in APs do not contain if-then-else clauses. These constructs can be handled

concrete type	abstract type	abstraction ψ
B	B	$\psi(s).v = s.v$
P_n	$\widehat{\mathsf{P}}_k$	$\psi(s).v = \widehat{\psi}_k(s.v)$
array $[\mathsf{P}_n]$ of B	array $[\mathsf{P}_k]$ of B	$\forall i \in \mathsf{P}_k \,:\, \psi(s).v[i] = s.v[i]$
array $[\mathsf{P}_n]$ of P_n	array $[\mathsf{P}_k]$ of $\widehat{\mathsf{P}}_k$	$\forall i \in \mathsf{P}_k \,:\, \psi(s).v[i] = \widehat{\psi}_k(s.v[i])$

TABLE I

The abstract state space $S_{\mathcal{A}}$ and the abstraction function $\psi : S(n) \to S_{\mathcal{A}}$. For a system variable v and $s \in S(n)$, the leftmost column gives the type of v in the concrete domain, the second column gives the type of v in $S_{\mathcal{A}}$, and the third column specifies the value v is assigned by $\psi(s)$ in terms of $s.v$.

by a straightforward splitting into multiple rulesets. The Murφ systems for German and FLASH are admissible, and from this experience, we believe that the systems for many other symmetric protocols will be admissible or easily modified to produce an admissible equivalent.

B. Abstraction

Let $\mathcal{P}(n) = (S(n), I(n), T(n))$ be a the denotation of an AP, P. We want to construct a mixed-abstraction, $\mathcal{A} = (S_{\mathcal{A}}, I_{\mathcal{A}}, O, U)$. In this section, we show how $S_{\mathcal{A}}$, $I_{\mathcal{A}}$, and O can be readily by syntactic transformations of the source-code of P. Section V extends this approach to the construction of U. To create these abstractions, we introduce a new type to represent type P_n from the concrete system; this type requires the user to choose a constant k. It is assumed throughout that k is *at least* the greatest number of ruleset parameters for any ruleset in P (typically, $k \leq 3$). Let $\widehat{\mathsf{P}}_k = \mathsf{P}_k \cup \{Other\}$, and given $x \in \mathsf{P}_n$, let $\widehat{\psi}_k(x) = x$ if $x \leq k$; otherwise, $\widehat{\psi}_k(x) = Other$. Table I specifies how each variable of $\mathcal{P}$ is typed in $\mathcal{A}$ and how the abstraction function ψ acts on v. Intuitively, $\psi(s)$ preserves B variable values, replaces values of type P_n greater than k with $Other$, and restricts arrays to the indices P_k (hence all array entries $v[i]$ for $i > k$ are deleted by ψ). Although ψ is a function, we will treat it as a relation, $\psi \subseteq S(n) \times S_{\mathcal{A}}$, and freely employ its inverse $\psi^{-1} \subseteq S_{\mathcal{A}} \times S(n)$. We call elements of P_k *non-abstracted* and elements of $\mathsf{P}_n \backslash \mathsf{P}_k$ *abstracted*. For every ruleset parameter i interpreted as abstracted, all updates with $a_{\mathsf{B}}[i]$ or $a_{\mathsf{P}}[i]$ appearing on the LHS are deleted. All instances $a_{\mathsf{B}}[i]$ or comparisons depending on $a_{\mathsf{P}}[i]$ appearing positively in the guard that depend replaced with $true$; those appearing negatively are replaced with $false$. Instances of i appearing on the RHS of assignments are replaced with $Other$. Finally, equality comparisons with i appearing positively in the guard are replaced with $true$. The state variables of $\mathcal{A}$ have the same names as those of $\mathcal{P}$, with the types changed as shown in Table I.

We now overload ψ to map rules of $\mathcal{P}$ system to rules that generate the state transitions of O. Rules of $\mathcal{P}$ that are not in rulesets are copied without change of syntax (therefore, with the implied change of types), to O. If ruleset $r : \rho \twoheadrightarrow \boldsymbol{a}$ depends on m ruleset parameters, consider the set of *rule instantiations*, obtained from assigning each ruleset parameter a value in P_n. This set is partitioned as $R_1, ..., R_{2^m}$, where all rule instantiations of R_j have the same partitioning of ruleset parameters into F and NF, where $i \in F \Leftrightarrow i \in \mathsf{P}_k$ and $i \in NF \Leftrightarrow i \in \mathsf{P}_n \setminus \mathsf{P}_k$ (since there are m ruleset parameters, there are 2^m possible partitions). Each set R_j abstracts to an abstract ruleset $\widehat{r}_j$ according to the described syntactic transformation. We denote the set of corresponding abstract rulesets to concrete ruleset r by $\psi(r) = \{\widehat{r}_1, ..., \widehat{r}_{2^m}\}$. Let $\widehat{r}_v : \widehat{\rho}_v \twoheadrightarrow \widehat{\boldsymbol{a}}_v$ denote the unique element of $\psi(r)$ such that all ruleset parameters are non-abstracted. Note that although the set of rule instantiations differ depending on the value of n, the set $\psi(r)$ does not, for any $n > k$, hence we can fix $n = k + 1$ to perform this abstraction.

Example: Consider the concrete rule from German `SendGntE`.

```
ruleset i : NODE do rule "SendGntE"
CurCmd = ReqE ∧ CurPtr = i ∧ Chan2[i].Cmd = Empty
∧ ¬ExGntd ∧ forall j : NODE do ¬ShrSet[j] end ==>
Chan2[i].Cmd := GntE; ShrSet[i] := true;
ExGntd := true; CurCmd := Empty;
```

The abstraction contains two corresponding rulesets, one where i is non-abstracted and one where i is abstracted, with the former corresponding to $\widehat{r}_v$:

```
ruleset i : NODE do rule "ABS_SendGntE1"
CurCmd = ReqE ∧ CurPtr = i ∧ Chan2[i].Cmd = Empty
∧ ¬ExGntd ∧ forall j : NODE do ¬ShrSet[j] end ==>
Chan2[i].Cmd := GntE; ShrSet[i] := true;
ExGntd := true; CurCmd := Empty;

rule "ABS_SendGntE2"
CurCmd = ReqE ∧ CurPtr = Other ∧ ¬ExGntd
∧ forall j : NODE do ¬ShrSet[j] end ==>
ExGntd := true; CurCmd := Empty;
```

V. Verifying Universal Quiescence

We want to verify properties of the form

$$\mathcal{P}(n) \models \mathsf{AG}\,\mathsf{EF}\,Q_n\ , \tag{3}$$

where $\mathcal{P}(n)$ is a parameterized system and

$$Q_n \quad = \quad G \wedge \textstyle\bigwedge_{i \in \mathsf{P}_n} L[i] \tag{4}$$

is the quiescence property to be verified. Here, G is a *boolean predicate*, meaning G only depends on variables of type B, while L is a local boolean predicate (defined in Sect. IV-A). To verify (3), we construct a mixed abstraction, $\mathcal{A} = (S_{\mathcal{A}}, I_{\mathcal{A}}, O, U)$, and that for all O-reachable states, there exists a U-path to a state that satisfies Q_n. To do so, we must address two key issues. First, Q_n cannot be established directly from $\mathcal{A}$, as Q_n refers to variables of the concrete system that do not appear in the abstraction. This is the "abstract quiescence insufficiency" defined in Section III-B, and Section V-A shows how it can be addressed. Second, U may omit transitions that are required to reach states that satisfy Q_n. This is the UA insufficiency from Section III-B, and we address it in Sections V-B and V-C.

A. Universally Quantified Quiescence

To show (3), we need to show that $L[i]$ holds for all i, not just non-abstracted i. Intuitively, we show that $\mathcal{A}$ can reach a state where $L[i]$ holds for all non-abstracted i, then use

Lemma 3 to exchange any abstracted j for which $L[j]$ might not hold with a non-abstracted i, find a path that establishes $L[i]$, and we can establish (3) by induction. For this approach to work, we must show that if G holds, then for each non-abstracted node i, there is a U-path to a state that satisfies $L[i]$ whose concretization in $\mathcal{P}(n)$ does not falsify $L[j]$ for any abstracted node j. To do this, we introduce the notion of L-preserving transitions.

For local boolean predicate L, abstract transition (s, s') is *L-preserving* if

$$\begin{aligned} & \forall w \in \psi^{-1}(s).\ \exists w' \in \psi^{-1}(s').\ w \rightsquigarrow_T w' \\ \wedge \quad & \forall i \in \mathsf{P}_n \setminus \mathsf{P}_k.\ w \in L[i] \Rightarrow w' \in L[i]\ . \end{aligned}$$

Abstract ruleset $\widehat{r}$ is L-preserving if all transitions in $[\![\widehat{r}]\!]$ are L-preserving. A mixed abstraction is called L-preserving if its UA transitions contain only L-preserving rules.

We can now state our main theorem for showing universally quantified quiescence; for the proof of Theorem 1 see the appendix.

Theorem 1 (Universally Quantified Quiescence): Let G denote a boolean predicate, L a local boolean predicate, and let $\mathcal{A}$ and $\mathcal{B}$ be mixed abstractions of $\mathcal{P}(n)$, and assume that $\mathcal{B}$ is L-preserving. If

1) $\mathcal{A} \models \mathsf{AG}\,\mathsf{EF}\,(G)$, and
2) $\mathcal{B} \models \mathsf{AG}\,(G \rightarrow \mathsf{EF}\,(G \wedge \bigwedge_{i \in \mathsf{P}_k} L[i]))$

then $\mathcal{P}(n) \models \mathsf{AG}\,\mathsf{EF}\,(G \wedge \bigwedge_{i \in \mathsf{P}_n} L[i])$.

B. Abstract Rule Tags

Given a program P, we use the syntactical abstraction technique to produce an abstract program $\widehat{P}$. In the remainder of this paper, we use the term "ruleset" to refer to Murφ-style rulesets with any degree of ruleset nesting including no such quantification – i.e. a "ruleset" could be a simple rule. We want to identify which rulesets of $\widehat{P}$ have denotations that are UA, and which are L-preserving, for a given local boolean predicate L. Throughout the rest of the paper we use $r : \rho \twoheadrightarrow \boldsymbol{a}$ and $\widehat{r}_v$ as defined in Section IV-B, and use $\widehat{r}_j$ to denote an arbitrary element of $\psi(r)$.

We will tag abstract rulesets tags from the following set $\{\mathsf{AUG}, \mathsf{AEG}, \mathsf{AUC}, \mathsf{AEC}\}$; the first two elements are called *guard tags*, and the last two are called *command tags*. These indicate reasons (in the guard and command, respectively) why the abstract ruleset is not trivially UA or L-preserving. An abstract ruleset can be tagged with *any* of the 16 subsets of these tags. AUG and AUC indicate that a universal quantifier has been abstracted; similarly AEG and AEC indicate that existential information has been abstracted.[3]

We call ρ and $\widehat{\rho}_j$ *syntactically equivalent* (SE) if they are expressed with identical syntax, and ρ contains no forall conditions. In this case, we attach no guard tags to $\widehat{r}_j$. Likewise, if $\boldsymbol{a}$ and $\widehat{\boldsymbol{a}}_j$ have identical syntax and contain no forall updates, then we attach no command tags. If a ruleset r has no guard or command tags, then it is simple to show that r is both UA and L-preserving for any local predicate L. Typically, the set of all such rulesets is insufficient to establish the desired quiescence property.

[3] Note that AEG and AEC don't indicate explicit existential quantifiers in the concrete system syntax; *existential* refers to the quantifier in the ruleset denotation, which ranges over the ruleset parameter.

Tag \Property	UA	L-preserving
AEG	Heuristic 1	Heuristic 2
AUG	Heuristic 3	Heuristic 4
AEC	None	Heuristic 2
AUC	None	Heuristic 4

TABLE II
Obligations associated with each ruleset tag and property pair. "None" means there is no obligation to show. A ruleset with no *tags* is L-preserving, while one with no *guard tags* is UA.

The elements of $\psi(r) \setminus \widehat{r}_v$ may not have SE guards because some ruleset parameter i is abstracted, so the abstraction will syntactically change the guard (except for degenerate cases). These rulesets have guards that optimistically abstract away references to abstracted i; this is safe when constructing the OA but not the UA. Such rulesets are tagged with AEG (abstract existential in guard).[3] When ρ contains a forall condition, it is necessarily weakened in every rule of $\psi(r)$. In this case, every ruleset of $\psi(r)$ including $\widehat{r}_v$ is tagged with AUG (abstract universal in guard).

Similarly, existential or universal updates may be missing from the command of an abstract ruleset, relative to the concrete version. If local update $a_b[i] := e_b$ appears in $\boldsymbol{a}$ for ruleset parameter i, then any ruleset of $\psi(r)$ where i is abstracted (that is, where the update $a_b[i] := e_b$ vanishes), is tagged with AEC (abstract existential command).[3] If $\boldsymbol{a}$ contains a forall update, then every ruleset of $\psi(r)$ is tagged with AUC (abstract universal command).

Example: Referring to the example in Section IV-B, abstract ruleset `ABS_SendGntE1` is tagged with AUG and no command tags, and abstract ruleset `ABS_SendGntE2` is tagged with AUG, AEG and AEC.

C. Heuristics

Each tag assigned to a ruleset corresponds to a set of proof obligations for showing it is UA or L-preserving (for some local boolean predicate L). For either of these properties, each tag must be separately *discharged* through the corresponding heuristic according to Table II. Once a tag is discharged we may safely ignore it as a potential reason why the desired property does not hold. Each of the heuristics involves model checking a mixed abstraction. In this Section, the various heuristics are stated; see the supplementary material [25] for proofs.

An abstract ruleset is called *local to* i (as a special case of having no tags) when the guard only depends on variables of type B, P, and the i^{th} index array variables a_{B}, and the command only updates the local state of non-abstracted i. Here, given an abstract or concrete state, the *local state* of i is simply the values of all array variables at index i. The transitions that compose such rules are called *local transitions*. A mixed abstraction with UA set U composed only of rulesets local to i is denoted $\mathcal{A}_{\ell(i)}$. Assuming ruleset $\widehat{r}$ is UA, we write

$\mathcal{A}_{\ell(i)} \models_{\widehat{r}} \mathsf{AG}(A \rightarrow \mathsf{EF}\, B)$ when every O-reachable A-state has a path to some B-state consisting of transitions of rules local to i and *necessarily a single transition* of ruleset $\widehat{r}$.

When showing rulesets are UA (Heuristics 1 and 3), note that the tags AEG or AUG indicate guards that are OA because they have abstracted away information about abstracted nodes. Our heuristics compute O-reachable states and exploit the path symmetry of Lemma 3 to find the possible local state of abstracted nodes under some boolean predicate. Then, if the local state of node i does not have a required property, we find "hidden paths" composed entirely of rulesets local to i that reach a state that does have the property. This assures that although some states in the concretization of abstract guard $\widehat{\rho}_j$ *do not* satisfy the corresponding concrete guard ρ, there is a guaranteed path that is not observable in the abstract system from every $\psi^{-1}(\widehat{\rho}_j)$ to a ρ-state. For simplicity, we present our heuristics for rulesets with at most one abstracted ruleset parameter, however generalizing is straightforward.

When showing rulesets are L-preserving it must be checked that aspects of the guard and update that have been abstracted away do not affect L-preservation in the abstracted nodes; Heuristics 2 and 4 pertain to this check. The obligations for these heuristics require that a certain transition must fire on each path that justifies the deadlock freedom property. Intuitively, when the heuristic obligation holds, the concrete paths that justify the tagged ruleset in question $\widehat{r}$ to be UA must have a certain form. For abstracted node i, each path is

- a (possibly empty) path composed of transitions of rules local to i, followed by
- a transition of concrete rule r (possibly changing non-local variables), followed by
- a (possibly empty) path composed of transitions of rules local to i.

Furthermore, we only seek a path when the starting state is an $L[i]$-state, and the final state must also be a $L[i]$-state. For which i this is shown depends on the heuristic. Heuristic 2 reasons about those abstracted nodes that are abstracted ruleset parameters in $\widehat{r}$. Heuristic 4 reasons about those abstracted nodes that are *not* abstracted ruleset parameters in $\widehat{r}$. Note that we assume a ruleset has been proven UA before it is proven L-preserving.

A few definitions are needed for the heuristic statements. If $\widehat{r}$ is an abstract ruleset with ruleset parameter i, let $\widehat{r}|_{i=1} : \widehat{\rho}|_{i=1} \rightarrow \widehat{\boldsymbol{a}}|_{i=1}$ be the ruleset where all instances of i are replaced with the constant value 1. Also, let $relax(i, \widehat{r})$ be the rule $\widehat{r}$ but with the values of variables $a_{\mathsf{B}}[i]$ and $a_{\mathsf{P}}[i]$ unconstrained in the guard. If $A \subseteq S_{\mathcal{A}}$, let $\Gamma(A)$ denote the strongest boolean predicate implied by A.

Heuristic 1: For ruleset $\widehat{r}_j$ tagged with AEG and with abstracted ruleset parameter i, suppose that $\widehat{r}_v$ is UA. If $\mathcal{A}_{\ell(1)} \models \mathsf{AG}(relax(\widehat{\rho}_v, i)|_{i=1} \rightarrow \mathsf{EF}(\widehat{\rho}_v|_{i=1}))$ then tag AEG is discharged for showing $\widehat{r}_j$ to be UA.

Heuristic 2: For ruleset $\widehat{r}_j$ tagged with AEG and/or AEC with abstracted ruleset parameter i, suppose that $\widehat{r}_v$ is UA. If $\mathcal{A}_{\ell(1)} \models_{\widehat{r}_v} \mathsf{AG}((relax(\widehat{\rho}_v, i)|_{i=1} \wedge L[1]) \rightarrow \mathsf{EF}(L[1]))$ then AEG and AEC are discharged for showing $\widehat{r}_j$ to be L-preserving.

Heuristic 3: For ruleset $\widehat{r}_j$ tagged with AUG, and let $\forall i \in \mathsf{P}_n$. $C[i]$ be the forall condition of ρ. If $\mathcal{A}_{\ell(1)} \models \mathsf{AG}(\Gamma(\widehat{\rho}_j) \rightarrow \mathsf{EF}(C[1]))$, then tag AUG is discharged for showing $\widehat{r}_j$ to be UA.

Heuristic 4: For ruleset $\widehat{r}_j$ tagged with AUG and/or AUC, let $\widehat{r}* \in \psi(r)$ be the abstract ruleset where all ruleset parameters of r are abstracted. If $\mathcal{A}_{\ell(1)} \models_{\widehat{r}*} \mathsf{AG}((\Gamma(\widehat{\rho}_j) \wedge L[1]) \rightarrow \mathsf{EF}(L[1]))$, then tags AUG and AUC are discharged for showing $\widehat{r}_j$ to be L-preserving.

We apply each of these heuristics by performing model checking using a mixed abstraction that uses *only* local rules for U. As local rules are identified entirely by syntax, they are known *a priori*; therefore, we could take a brute force approach that attempts to use our heuristics to prove every abstract rule is UA and L-preserving. However, we prefer to take a counter-example driven approach, as there are two distinct situations in which our heuristics may not suffice that arose in our case studies. Firstly, additional auxiliary variables may be needed to capture the system state with a slightly finer-grained abstraction. Secondly, if the ruleset is not underapproximate, manual guard strengthening or splitting into multiple rulesets may help. These are illustrated with examples in Section VI.

VI. Case Studies

Mixed abstractions are expressed as Murφ models. The OA rulesets are borrowed from Chou *et al.* [6] and the (initial) UA transitions are derived manually according to tags – those rules with no guard tags. Thus, the UA rulesets are maintained as a subset of the OA rulesets. Rulesets with no tags at all are identified as L-preserving, and the relevant subset of these are identified as local.

We use a distributed explicit state model checker for Murφ called PReach [26] for the mixed abstraction checks. Initially designed to check state-invariants, we have added a feature to check CTL properties of the form $\mathsf{AG}(p \rightarrow \mathsf{EF}\, q)$. The search algorithm is simple: for every $(p \wedge \neg q)$-state s visited during the forward reachability computation, choose an enabled rule of U and fire it to reach a new state. Firing rules of U continues until one of the following occurs. 1) a q-state is found, 2) a U-dead-end state is found, or 3) a cycle is detected. In the first case, a path from s to a q-state exists and we proceed with the forward reachability computation. In the second case, there *may* not exist such a path (although we believe that in practice this is strong evidence that no path exists). If a cycle is found, this is usually an indication that U contains rules that do not help us reach q-states, so we might as well exclude them and try again[4]. For example, there are several easily identifiable rules in both German and FLASH that initiate requests by injecting messages, and are not useful transitions in finding a quiescent state where all messages are consumed. Notice that deadlock freedom properties can be verified by a CTL model

[4]Removing transitions from U trivially preserves mixed-abstractions.

checker, but for our case studies we chose PREACH because it was straightforward to implement the notion of UA rulesets and counterexample generation.

Due to space constraints, this section contains a brief overview of the case studies. For a more detailed report, the reader may refer to supplementary material [25] including the Murφ sources.

A. Automatic Deadlock Freedom Predicates

As mentioned above, it is common when checking antecedent 1 of Theorem 1 to reach a U-dead-end state $\check{s}$ where no further progress can be made toward the goal. When this occurs, the model checker reports a failure and prints the rules of O that are enabled in $\check{s}$, as a guide to the user of which rules could be useful to prove UA and add to U. These enabled rulesets necessarily have tags AEG or AUG or both. We have written a simple tool that, given a particular rule/ruleset name, will determine the tags and generate the model checking obligation to prove it is UA through Heuristics 1 and 3.

Example: Suppose we seek to show ruleset $\widehat{r}_2$ = `ABS_SendGntE2` is UA, and suppose it is already known by Heuristic 3 that associated $\widehat{r}_1 = \widehat{r}_v$ = `ABS_SendGntE1` is UA. Ruleset `ABS_SendGntE2` is tagged with AEG because ruleset parameter i is abstracted. Then, $relax(\widehat{\rho}_v)|_{i=1}$ is

```
CurCmd = ReqE ∧ CurPtr = 1 ∧ ¬ExGntd
∧ forall j : NODE do ¬ShrSet[j] end
```

and $\widehat{\rho}_v|_{i=1}$ is

```
CurCmd = ReqE ∧ CurPtr = 1 ∧ Chan2[1].Cmd = Empty
∧ ¬ExGntd ∧ forall j : NODE do ¬ShrSet[j] end
```

As implemented, our tool does not support automatic generation of the properties to check for Heuristics 2 and 4. However, this is generally straightforward to do by hand, and could be automated as well. In cases when the deadlock freedom property for some Heuristic when applied to ruleset $\widehat{r}_j$ fails to verify, the user may use the counterexample trace as a guide for strengthening $\widehat{\rho}_j$ manually. Any ruleset of O may be duplicated and strengthened with some predicate, which is trivially sound because the O transitions are not changed. The resulting strengthened ruleset might satisfy the Heuristic deadlock freedom property and be proven UA or L-preserving. Such manual strengthening is required in the verification of both German and FLASH.

B. The German Protocol

The system used for O is the abstract Murφ model for German of Chou *et al.*, instantiated with a single non-abstracted node ($k = 1$). The initial set of UA transitions U_0 includes all rulesets with no guard tags and the local subset of these are also identified.

The property we verify is (4), where G states that that the directory is not currently processing a transaction (`CurCmd = Empty`) and $L[i]$ states that all communication channels associated with the i^{th} cache are empty:

```
 Chan1[i].Cmd = Empty ∧ Chan2[i].Cmd = Empty
∧ Chan3[i].Cmd = Empty.
```

Antecedent 1 of Theorem 1 requires $\mathcal{A} \models \mathsf{AG}\,\mathsf{EF}\,(G)$ for a mixed-abstraction $\mathcal{A}$. Checking this property, the model-checker gets stuck at a U-dead-end state where the rule `ABS_SendGntE1` is enabled (see Section IV-B). Our tool recognizes this as a AUG-tagged rule and generates the obligations to according to Heuristic 1 so the rule can be soundly added to U. The model checker discharges the obligation, and `ABS_SendGntE1` is added to U.

Checking Antecedent 1 is repeated with the weakened U and gets stuck three more times: once where `ABS_SendGntE2` is enabled (tagged with AEG *and* AUG), and twice where other AEG-tagged rulesets are enabled. The Heuristic 3 obligation for `ABS_SendGntE2` is identical to the one previously shown for `ABS_SendGntE1`, so there is no need to repeat its verification. The tool generates the Heuristic 1 obligation and it is discharged by model checking. In the other two, AEG cases, the corresponding rulesets $\widehat{r}_v$ are already known to be in U, so we proceed directly with the tool and obligations for Heuristic 1 are generated. One is discharged automatically; the other requires human guidance because the generated deadlock freedom property fails to verify. An examination of the counterexample reveals that when exclusive access has been granted to an abstracted node, there is no pointer indicating which node has been granted (only a flag to indicate that it has indeed been granted, `ExGntd`). Without this pointer, the permutation of Heuristic 1 is not applied to the proper abstract node actually holding exclusive access. Although manual, the solution is straightforward: add a new system variable `EPtr` of type P that points to the node holding exclusive access, and strengthening the guard of the ruleset. This is done in a sound manner where only the ruleset version we prove is UA is strengthened in this way; the original ruleset belonging to O is not modified. After this modification, the relevant property is verified.

Having added these four rules to U of mixed abstraction $\mathcal{A}$, Antecedent 1 of Theorem 1 is established by model checking. We now describe the procedure to show of Antecedent 2. Initially, every ruleset with no tags are known to be L-preserving are added to U for mixed abstraction $\mathcal{B}$. Model checking then reveals that two additional rules are needed to establish the Antecedent: `ABS_SendGntE1` (tagged AUG) and `ABS_RecvInvAck2` (tagged AEG and AEC). These tags are discharged by automatically generating and checking the obligations of Heuristics 4 and 2, respectively. Adding these two rules to U for mixed abstraction $\mathcal{B}$ allows Antecedent 2 to hold and completes the verification of the German protocol.

C. The FLASH Protocol

The quiescence property verified of FLASH is of the same form as (4), and states that all channels are clear, and the directory is not waiting to perform a write-back[5]. Antecedent 1 of Theorem 1 holds immediately using the initial set of UA rulesets having no guard tags.

[5]Although the Murφ system for the mixed abstraction of FLASH contains rules where two index variables have been instantiated as *Other*, none of these must be shown UA/L-preserving to prove our example property. Some such rules are needed to be shown UA if the conjunct `¬Pending` is added to the quiescent property. We omit these from this paper for ease of presentation, but note that similar reasoning to Heuristic 1, which assumes only one such index variable, is sufficient.

To show Antecedent 2 of Theorem 1, we start with the set U of L-preserving states provided by tag examination and use model checking as with the German protocol. Four rules, each tagged with AEG and AEC, must be shown L-preserving. We first show that they are UA, by applying Heuristic 1. For two of these rulesets, model checking the obligations for Heuristic 1 succeeds. For the other two, model checking fails upon reaching a dead-end state $\check{s}'$ where no local rules to 1 are enabled. The manual strengthening needed in for these two ruleset is identical; without loss of generality let the ruleset be $\widehat{r}_j$. Inspecting the counter example reveals that the state $s \in relax(\widehat{r}_v, i)|_{i=1}$ that led to $\check{s}'$ has different values for some B-type variables that those in $\check{s}$, the original dead-end state revealed when checking Antecedent 2, where $\widehat{r}_j$ is enabled. This indicates that the guard $\widehat{r}_j$ is too weak and must be strengthened with a predicate on these variables. We duplicated the ruleset for the aforementioned reasons of soundness, and strengthened the guard with a predicate requiring these variables to match their value in $\check{s}$. Then, the automated procedure completed successfully and the four rules are established as UA. To show they are L-preserving, Heuristic 2 is applied to each ruleset and the obligations are discharged automatically; this establishes the quiescence property by Theorem 1. With regard to the manual strengthening step, we note that in principle the model checker could classify the reachable states of $relax(\widehat{r}_v, i)|_{i=1}$ for which a path to $\widehat{r}_v|_{i=1}$ is found versus those where no such path is found. Thus, the strengthening predicate could be generated automatically.

VII. Discussion and Future Work

We presented a practical method for proving deadlock freedom in parameterized cache coherence protocols. Our approach uses a mixed abstraction of over (OA) and under (UA) approximate transitions to parametrically verify properties of the form $\mathcal{P}(n) \models \mathsf{AG}\,\mathsf{EF}\,G \wedge \bigwedge_{i \in \mathsf{P}_n} L[i]$, where n is the number of cache *nodes*, G is a predicate depending on boolean variables, and L is a predicate depending on boolean variables local to each node i. We infer this parameterized property by model checking a pair of antecedent *deadlock freedom* properties in an automatically generated mixed abstraction.

First, the model checker explores all states s reachable through OA transitions, and for each s a UA-path to some G-state s' constructed via a forward search. When no such s' is found, the user determines if s is only reachable due to the overapproximation of OA, or if s' is unreachable from s due to UA being too strong. For the former, we use existing methods to strengthen OA. For the latter, we have presented heuristic methods to soundly weaken the UA transitions by proving some transitions of OA are in fact UA. These heuristics involve checking specific deadlock freedom properties in the mixed abstraction. Second, it is verified that all G-states reachable through OA transitions have an *L-preserving* UA-path to a state where G holds *and* $L[i]$ holds for all k nodes maintained by the abstraction. Abstract transitions that are L-preserving have the property that the set of corresponding paths in the concrete system will *preserve* $L[i]$ for all nodes i that are abstracted away. Once again, we provide heuristic methods for showing that OA transitions are in fact L-preserving UA transitions. With each of these antecedents established, a simple induction proof is used to show the parameterized deadlock freedom property holds.

For the German and FLASH protocols, the strengthening of OA that was required to prove safety [6] was sufficient to establish liveness as well. Furthermore, most weakenings needed for UA were identified and verified without need for human insight. The only places where human reasoning was needed was to identify one auxiliary variable needed in the German model, and three guard strengthenings for FLASH. We believe that these steps are candidates for automation as well. Such automation may be desirable when these techniques are applied to more complicated protocols.

We described an enhancement to the PReach model checker [26] to support the "EF" part of our model checking obligations using a forward search to find the existential paths. Though the reachable state computation is fully distributed, the EF searches currently are not; one area of future work is to distribute this aspect of the model checking. However, our current implementation running with a single thread was sufficient to handle all the obligations for our two case studies. The largest model was an abstraction of FLASH that has about 2.4 M reachable states and, for the properties of our case study, each was checked on a modern desktop machine in less than 10 minutes.

As another direction of future work, we are in the process of writing a Murφ model of the L2 cache controller in the OpenSPARC multiprocessor design. Here the parameter of interest is memory addresses, rather than cache IDs. This is interesting since different addresses share resources in non-trivial ways that can lead to deadlock in our experience with real designs. Investigating parameterized deadlock freedom of this cache controller will test the applicability of our approach of a vastly different parameterized verification problem.

Acknowledgment

We thank John O'Leary for providing an Ocaml Murφ front-end that is the basis for our tag/heuristics tool, and anonymous reviewers for constructive feedback. This research was funded in part by NSERC RGPIN 138501-07, NSERC graduate fellowships, and a grant by Oracle Corporation.

References

[1] R. C. Holt, "Some deadlock properties of computer systems," *ACM Computing Surveys*, vol. 4, no. 3, pp. 179–196, 1972.

[2] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 1999.

[3] K. Apt and D. Kozen, "Limits for automatic verification of finite-state concurrent systems," *Information Processing Letters*, vol. 15, pp. 307–309, 1986.

[4] K. L. Mcmillan, "Verification of infinite state systems by compositional model checking," in *in CHARME*. Springer, 1999, pp. 219–233.

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

[6] C.-T. Chou, P. K. Mannava, and S. Park, "A simple method for parameterized verification of cache coherence protocols," in *FMCAD*, 2004, pp. 382–398.

[7] S. Krstic, "Parameterized system verification with guard strengthening and parameter abstraction." in *Automated Verification of Infinite-State Systems*, 2005.

[8] 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*, 2007, pp. 29–38.

[9] J. Bingham, "Automatic non-interference lemmas for parameterized model checking," in *Formal Methods in Computer Aided Design (FMCAD)*, 2008.

[10] Y. Li, "Mechanized proofs for the parameter abstraction and guard strengthening principle in parameterized verification of cache coherence protocols," in *Proceedings of the 2007 ACM symposium on Applied computing*, 2007, pp. 1534–1535.

[11] J. O'Leary, M. Talupur, and M. R. Tuttle, "Parameterized verification using message flows: An industrial experience," in *International Conference on Formal Methods in Computer Aided Design (FMCAD)*, 2009.

[12] D. Dams, R. Gerth, and O. Grumberg, "Abstract interpretation of reactive systems," *ACM Trans. Program. Lang. Syst.*, vol. 19, no. 2, pp. 253–291, 1997.

[13] Z. Manna and A. Pneuli, "Completing the temporal picture," *Theoretical Computer Science*, vol. 83, no. 1, pp. 97–130, 1991.

[14] K. L. McMillan, "Circular compositional reasoning about liveness," in *Correct Hardware Design and Verification Methods (CHARME)*, 1999, pp. 342–345, an extended version appeared as a Cadence technical report.

[15] K. McMillan, "Personal correspondence," 2011.

[16] Y. Fang, N. Piterman, A. Pnueli, and L. Zuck, "Liveness with invisible ranking," *Int. J. Software Tools for Technology Transfer*, vol. 8, no. 3, pp. 261–279, June 2006.

[17] A. Pnueli, J. Xu, and L. D. Zuck, "Liveness with (0, 1, infinity)-counter abstraction," in *Proceedings of the 14th International Conference on Computer Aided Verification (CAV)*, 2002, pp. 107–122.

[18] K. Baukus, S. Bensalem, Y. Lakhnech, and K. Stahl, "Abstracting WS1S systems to verify parameterized networks," in *International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, 2000, pp. 188–203.

[19] K. Baukus, Y. Lakhnech, and K. Stahl, "Parameterized verification of a cache coherence protocol: Safety and liveness," in *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, 2002, pp. 317–330.

[20] K. G. Larsen and B. Thomsen, "A modal process logic," in *Third Annual Symposium on Logic in Computer Science (LICS)*, 1988, pp. 203–210.

[21] R. Cleaveland, S. P. Iyer, and D. Yankelevich, "Abstractions for preserving all CTL* formulae," Tech. Rep., 1994, tech. Rep. 9403, Dept. of Comp. Sc., North Carolina State University, Raleigh, NC.

[22] N. Lynch and F. Vaandrager, "Forward and backward simulations – part I: Untimed systems," *Information and Computation*, vol. 121, no. 2, pp. 214–233, 1995.

[23] C. N. Ip and D. L. Dill, "Better verification through symmetry," in *11th IFIP WG10.2 International Conference on Computer Hardware Description Languages and their Applications*, 1993, pp. 97–111.

[24] 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*, 1992, pp. 522–525.

[25] B. Bingham, J. Bingham, and M. Greenstreet, "Supplementary material," `http://www.cs.ubc.ca/~binghamb/fmcad2011.html`, 2011.

[26] B. Bingham, J. Bingham, F. M. de Paula, J. Erickson, G. Singh, and M. Reitblatt, "Industrial strength distributed explicit state model checking," in *Parallel and Distributed Model Checking*, 2010.

APPENDIX

BNF for Admissible Parameterized Systems

AdmissibleSystem : r | ps ';' ar ;
r : rule | ruleset ;
rule : guard '==>' action ;
guard : '(' gTerm ')' | guard 'AND' '(' gTerm ')' ;
gTerm : '(' bTerm ')' | '(' eTerm ')' | '(' fTerm ')' ;
bTerm : bConst | bVar | bArray '[' pVar ']' | bTerm 'AND' bTerm
| bTerm 'OR' bTerm | | 'NOT' bTerm | '(' bTerm ')' ;
pComp : pTerm '=' pTerm ;
pTerm : pVar | pArray '[' pVar ']' ;
action : assignment | action ';' assignment ;
assignment : simpleAssignment | forAllAssign ;
simpleAssignment : bVar ':=' bTerm | bArray '[' pVar ']' ':=' bTerm
| pVar ':=' pTerm | pArray '[' pVar ']' ':=' pTerm ;
forAllAssign : 'forall' pVar ':' pType 'do' assignment 'end' ;
ruleset : 'ruleset' pVar ':' pType 'do' r 'end' ;

Where *bVar* is an identifier for a boolean-scalar variable, *bArray* is an identifier for a boolean-array variable, *pVar* is an identifier for a scalar variable of the parameter type, and *pArray* is an identifier for an array variable of the parameter type.

Restrictions: Any *pVar* declared as a ruleset index may appear in at most one *pComp* conjunct of any guard.

Proof of Theorem 1

Let us fix a mixed-abstraction $\mathcal{A} = (S_\mathcal{A}, I_\mathcal{A}, U, O)$ for $\mathcal{P}(n)$, where $n > k$. We also use L to denote a local boolean predicate, and $\mathcal{B} = (S_\mathcal{A}, I_\mathcal{A}, U_\mathcal{B}, O)$ to denote a mixed abstraction with only L-preserving transitions for $U_\mathcal{B}$. For $i, j \in \mathsf{P}_n$, define $\mathsf{P}_i^j \subseteq \mathsf{P}_n$ as $\{\ell : i \leq \ell \leq j\}$.
Let permutation $\pi_{j \leftrightarrow h}$ map elements of P_n according to

$$\pi_{j\leftrightarrow h}(i) = \begin{cases} j & \text{for } i = h, \\ h & \text{for } i = j, \\ i & \text{otherwise.} \end{cases}$$

Let T be shorthand for $T(n)$ and let $Reach$ denote the reachable states of $\mathcal{S}(n)$.

Theorem 1 (Universally Quantified Quiescence): Let G be a boolean predicate. If

1) $\mathcal{A} \models \mathsf{AG}\,\mathsf{EF}\,(G)$, and
2) $\mathcal{B} \models \mathsf{AG}\,(G \rightarrow \mathsf{EF}\,(G \wedge \bigwedge_{i\in\mathsf{P}_k} L[i]))$

then $\mathcal{P}(n) \models \mathsf{AG}\,\mathsf{EF}\,(G \wedge \bigwedge_{i\in\mathsf{P}_n} L[i])$.

Proof: For $1 \leq h \leq n$, let J_h denote the property $\forall w \in G \wedge Reach.\ \exists w' \in (G \wedge \bigwedge_{i\in\mathsf{P}_h} L[i])$ where $w \rightsquigarrow_T w'$, and $\forall i \in \mathsf{P}_{k+1}^n.\ w \in L[i] \rightarrow w' \in L[i]$.
By definition, Antecedent 2 implies J_k. Assume J_h holds for $k \leq h < n$. Applying permutation $\pi_{1\leftrightarrow h+1}$ to J_k gives $\forall w \in G \wedge Reach.\ \exists w' \in (G \wedge \bigwedge_{i\in\mathsf{P}^{h+1}} L[i])$ where $w \rightsquigarrow_T w'$, and $\forall i \in \mathsf{P}_{k+1}^n.\ w \in L[i] \rightarrow w'^2 \in L[i]$. This property with Antecedent 2 implies J_{h+1} by transitivity. Thus, property J_n follows by induction. The paths implied by Antecedent 1 composed with those of J_n complete the proof by transitivity. ■