Parametrized System Verification with Guard Strengthening and Parameter Abstraction

Sava Krstić [1,2]

Strategic CAD Labs
Intel Corporation
Hillsboro, Oregon, USA

Abstract

We give complete mathematial foundations for the method, recently developed by Chou, Mannava, and Park, for verifying safety properties of cache coherence protocols. The method employs a specific form of counterexample-guided abstraction refinement and is originally described on worked-out examples of the German and FLASH protocols. We describe and prove the method at an abstract level, thus establishing its scope and opening the way to its further mechanization.

Key words: Parametrized systems, verification, abstraction

1 Introduction

Descriptions of many systems of practical importance include an arbitrary parameter set: the set of agents in security protocols, the set of nodes in cache coherence protocols, etc. If $\mathcal{P}$ is such a parametrized system and $\mathcal{P}(n)$ denotes its instance where the parameter set has cardinality n, then there is a chain of simulations

$$\mathcal{P}(1) \longrightarrow \mathcal{P}(2) \longrightarrow \mathcal{P}(3) \longrightarrow \dots \tag{1}$$

so the safety properties true of $\mathcal{P}(n)$ are also true for all $\mathcal{P}(i)$ for $i < n$. Unfortunately but typically, the systems in this chain quickly become prohibitively large for straightforward model checking and the only way of proving correctness of, say, $\mathcal{P}(100)$ is by *parametrized verification*—giving a uniform correctness proof that works for all parameter set sizes n.

[1] Thanks to Ching-Tsun Chou, Jim Grundy, and Robert Jones for their comments. Thanks also to Yi Lv for pointing out an inaccuracy in an earlier version of Theorem 5.1.
[2] Email: sava.krstic@intel.com

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

It is often intuitively clear that all subtleties of behaviors of $\mathcal{P}$ are essentially present in $\mathcal{P}(m)$ for some small value m, and for some protocols it has been actually proved that there exists m such that the correctness of $\mathcal{P}(m)$ implies the correctness of $\mathcal{P}$. This is an important method, even though the class of protocols it currently covers and the upper bounds for the cut-off value m limit its scope. See [7,1] for this and [11,2,9] for alternative approaches to parametrized verification.

In a recent paper [4], Chou, Mannava, and Park offer a novel method inspired by McMillan's work [11], whereby the model-checking of a safety property ϕ in $\mathcal{P}(m)$ extends to a proof that ϕ holds in $\mathcal{P}$ for any choice of the parameter set. In Figure 1 and the following paragraph, we give a high-level description of this method; a full account is given in Section 5.

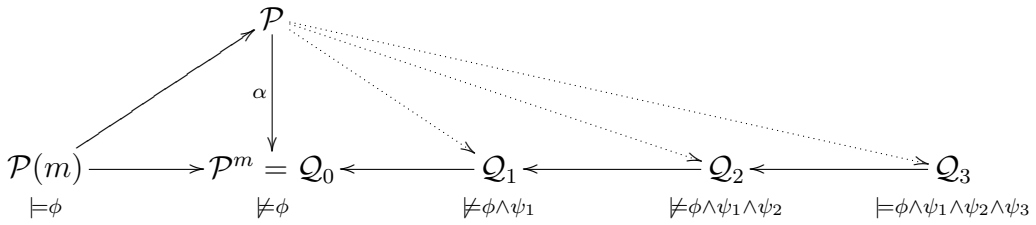

Fig. 1. Checking $\mathcal{P} \models \phi$ by the Chou-Mannava-Park Method. Solid arrows represent simulation relations, all except α being system inclusions. The method succeeds in this illustration after three iterations when it can be inferred that the dashed arrows are also simulations.

In its first phase, the method calls for construction of a *parameter abstraction* map α that replaces the parameter set $\{p_1, \ldots, p_n\}$ of $\mathcal{P}$, where n is arbitrarily large, by merging all but a few of its elements into a single abstract element. The "few" is a judiciously chosen small integer m. The resulting abstracted system $\mathcal{P}^m$ contains $\mathcal{P}(m)$ as a subsystem, but usually fails to satisfy the invariant ϕ. In the second phase, a sequence of progressively better approximations $\mathcal{Q}_1, \mathcal{Q}_2, \ldots$ of $\mathcal{P}$ is constructed, starting with $\mathcal{Q}_0 = \mathcal{P}^m$. The second phase follows the iterative "counterexample-guided abstraction refinement" paradigm [5,10], and the heart of the method is in the specifics of this refinement process. At each step, the analysis of the counterexample trace provided by the model checker results in: (1) locating a rule r of $\mathcal{P}$ and a corresponding abstracted rule r' of $\mathcal{Q}_i$ that caused a spurious transition in the trace; (2) finding a new purported invariant (called "non-interference lemma" after [11]) ψ_{i+1} of $\mathcal{P}$ which, if true, would allow us to strengthen the guard (precondition) of the rule r' so that the offending transition becomes impossible; (3) construction of $\mathcal{Q}_{i+1}$ by replacing r' with the strengthened rule. The proof is completed when some $\mathcal{Q}_k$ is found to satisfy ϕ and all the putative invariants $\psi_1, \ldots, \psi_k$.

The original paper [4] describes the method on the worked-out examples

of the German and FLASH cache coherence protocols, verified by the authors with the aid of the model-checker *Mur*φ [8]. The authors note apparently circular reasoning in their treatment of non-interference lemmas (assume them first to strengthen the rules of abstracted systems, then prove them for the so modified systems) that brings about the question of the method's own correctness. They address the issue by proving some germane general theorems about transition systems and using them to justify what is done in the verification process for the two concrete protocols.

Our goal in this paper is to describe the Chou-Mannava-Park method abstractly, without reference to examples, and to prove its correctness at the same abstract level. This seems to be a necessary first step towards further mechanization of the method—the highest desideratum in [4]. It also helps in determining the scope of the method, not easily discernible from the original paper. In large part, the method is a syntactic affair, and the matters syntactical are treated in [4] mainly by way of examples in the expressive language of *Mur*φ. The following questions need answers by means of precise definitions, and we supply them in this paper.

- How general can the system $\mathcal{P}$ be? (For example, the method allows system variables of array types $N \Rightarrow X$ and $N \Rightarrow N$, where N is the parameter type, but some restrictions must apply to the form of the system rules in which these variables appear.)
- What is the system transformation that produces $\mathcal{P}^m$ starting with $\mathcal{P}$?
- What is the system transformation that produces $\mathcal{Q}_{i+1}$ from $\mathcal{Q}_i$ and the non-interference lemma ψ_{i+1}?

We will show that the method's correctness rests on two simple statements. The first (Proposition 2.1) describes a guard strengthening technique for proving safety properties. It is an inductive principle interesting in its own right and it explains the apparently circular reasoning in [4]. The second statement (Theorem 4.2) simply says that the parameter abstraction map α is a simulation. This is taken as self-evident in [4] for the two examples considered, but in general requires proof which is not obvious. In fact, the syntactic restrictions on the form of rules in the system $\mathcal{P}$ are made so that this proof can go through.

The paper is organized as follows. In Section 2 we describe the syntax and semantics of systems and prove the guard strengthening principle. In Section 3 we define the syntax of *symmetric systems*, which is more primitive than *parametrized systems* (like those of *Mur*φ), but easier to reason about. Section 4 is about parameter abstraction. In Section 5 we prove the main Theorem 5.1 from Proposition 2.1 and Theorem 4.2. Then we describe the Chou-Mannava-Park method and derive its correctness from Theorem 5.1.

Technically, the paper is self-contained, but it lacks examples. We refer the reader to the first two sections of [4] for the necessary background.

The proofs are relegated to the Appendix.

2 Systems, Presentations, and Guard Strengthening

A *state transition system*, or just *system*, is a triple $(S, I, \rightarrow)$, where S is a set, I is a subset of S and $\rightarrow$ is a binary relation on S. The elements of S are the *states* of the system, those in I are the *initial states*, and $\rightarrow$ is the system's *transition relation*. A state $s \in S$ is *reachable* if there exists an *execution chain* $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_k$ with $s_0 \in I$ and $s_k = s$.

A function $h\colon S_1 \rightarrow S_2$ is a *simulation* from the system $(S_1, I_1, \rightarrow_1)$ to the system $(S_2, I_2, \rightarrow_2)$ if it maps I_1 to I_2 and $\rightarrow_1$-transitions to $\rightarrow_2$-transitions. Clearly, a simulation maps every reachable state of S_1 to a reachable state of S_2.

Systems will be described by *system presentations*, which are triples of the form $\mathcal{P} = (V, \iota, R)$, where

- V is a set of *system variables*, each associated with its *type*;
- ι is a predicate (the *initialization predicate*);
- R is the set of *transition rules*, each being a set of guarded parallel assignments

$$\rho \twoheadrightarrow \{v_1 := e_1 \ ; \ \ldots \ ; \ v_k := e_k\}, \tag{2}$$

  where ρ is a predicate, the v_i are distinct system variables, and each e_i is a term of the same type as v_i.

Furthermore, we assume the following conventions about the language of terms. Terms and are constructed using typed mathematical constants (operations of arbitrary arity, including individual constants) and variables from V, with function application being the only means to build larger terms from smaller ones. (Terms have no side effects.) Predicates are just terms of the boolean type. The equality operator for each type is given as a constant.

By definition, the subterms of a rule (2) are the subterms of $\rho, e_1, \ldots, e_k$.

As for the family of legitimate types, we assume it is generated by a set of *basic types* (booleans, naturals, perhaps more) using the set-theoretical product, disjoint sum, and function space constructors, denoted $\times, +, \Rightarrow$ respectively. We will write $\mathsf{type}(e)$ for the type of a term e.

Following *Mur*φ, we allow assignments occurring in the rules to be of the form $v_i := \mathsf{undefined}$. However, we will use a simpler semantics in which such a rule is just an abbreviation for the set of rules of the form $v_i := t$, for all elements t of the type of v_i.

Every presentation $\mathcal{P} = (V, \iota, R)$ determines a system $|\mathcal{P}| = (S, I, \rightarrow)$ in a well-known fashion, as follows. First, S is the set of all variable assignments—the maps that associate to each $v \in V$ an element of $\mathsf{type}(v)$. For every term e we write $s(e)$ for the value of e under the assignment s, and we also write $s \models \phi$ to mean $s(\phi) = \mathsf{true}$. By definition, the set I consists of the states $s \in S$ such that $s \models \iota$. By definition, $s \rightarrow s'$ holds if there exists a rule $\rho \twoheadrightarrow \{v_1 := e_1 \ ; \ \ldots \ ; \ v_k := e_k\}$ such that $s \models \rho$ and s' is the assignment given by $s'(v_i) = s(e_i)$ for $i = 1, \ldots, k$ and $s'(v) = s(v)$ for $v \neq v_1, \ldots, v_k$. We say

that a predicate ϕ is an *invariant* of $\mathcal{P}$ and write $\mathcal{P} \models \phi$ when $s \models \phi$ holds for all reachable states of S.

In order to prove that ϕ is an invariant of a system $\mathcal{P}$, we can first strengthen the rules of $\mathcal{P}$ by adding ϕ to their guards, and then prove that ϕ is an invariant of the newly restricted system. This *guard strengthening* principle is justified by the following proposition.

Proposition 2.1 *Let ϕ be a predicate over variables of the system $\mathcal{P}$. Let $\mathcal{P}^\#$ be the system obtained by replacing each rule*[3] *$\rho \rightarrowtriangle \boldsymbol{a}$ of $\mathcal{P}$ with $\rho^\# \rightarrowtriangle \boldsymbol{a}$, where $\rho^\#$ is any logical consequence of $\phi \wedge \rho$. Then $\mathcal{P}^\# \models \phi$ implies $\mathcal{P} \models \phi$.*□

3 Symmetric and Parametrized Systems

Let us fix n and a finite *parameter set* $N = \{p_1, \ldots, p_n\}$. We assume that N is a basic type in the family of types used to specify our systems, and that $p_1, \ldots, p_n$ are constants of type N. If π is a permutation of N and e is an arbitrary term, we denote by $\pi(e)$ the term obtained by replacing all occurrences of each constant p_i with $\pi(p_i)$. We will say that e is a *symmetric* term if $\vdash \pi(e) = e$ for every permutation π. Since predicates are viewed as terms of boolean type, we have that a predicate ϕ is symmetric when $\vdash \pi(\phi) \iff \phi$ for every π. The conjunction $\phi_{sym} = \bigwedge_\pi \pi(\phi)$ taken over all permutations π is the weakest symmetric predicate that implies ϕ.

We say that $\mathcal{P} = (V, \iota, R)$ is a *symmetric presentation* when

- ι is a symmetric predicate;
- For every permutation π and rule $\rho \rightarrowtriangle \{v_1 := e_1 \ ; \ \ldots \ ; \ v_k := e_k\}$ of R, there exists a rule in R equivalent to $\pi(\rho) \rightarrowtriangle \{v_1 := \pi(e_1) \ ; \ \ldots \ ; \ v_k := \pi(e_k)\}$.[4]

Lemma 3.1 *If $\mathcal{P}$ is a symmetric presentation and $\mathcal{P} \models \phi$, then $\mathcal{P} \models \phi_{sym}$.*□

If $\mathcal{P}$ is a symmetric presentation, then there is an induced action of the permutation group of N on the system $|\mathcal{P}|$, i.e. this group is a symmetry group of $|\mathcal{P}|$, as defined in [6].

One can give compact descriptions of symmetric systems by extending the system presentation language with a set of fresh *auxiliary variables* that have the parameter type N and can be quantified over. The quantified predicates $(\forall u)\phi$ and $(\exists u)\phi$ are just abbreviations for $\bigwedge_{i=1}^n \phi[p_i/u]$ and $\bigvee_{i=1}^n \phi[p_i/u]$. Similarly, a quantified array assigment $(\forall u) f[u] := a$ is an abbreviation for the set of assignments $f[p_i] := a[p_i/u]$ $(i = 1, \ldots, n)$. (See Section 4 for array assignments.) Finally, a quantified rule $(\forall u) r$ abbreviates the set of rules $r[p_i/u]$ $(i = 1, \ldots, n)$. We define a presentation with auxiliary variables and quantification over them in predicates, assignments, and rules to be a *parametrized*

[3] The notation $\boldsymbol{a}$ in $\rho \rightarrowtriangle \boldsymbol{a}$ is for a set of assignments, as in (2).

[4] Two rules are equivalent if their guards are logically equivalent ($\vdash \rho \iff \rho'$) and for each assignment $v := e$ occurring in one of them there is an assignment $v := e'$ in the other such that $\vdash e = e'$.

presentation if no constant of type N occurs explicitly in it.

It is easy to see that a parametrized presentation is equivalent to a symmetric presentation (without occurrences of auxiliary variables). More precisely, every parametric presentation defines a family of symmetric presentations, one up to isomorphism for each possible cardinality of the parameter set.

Model checkers that employ symmetry reduction techniques, like *Mur*φ and SMV, enforce specifying symmetric systems by parametric presentations. In this paper, we use symmetric presentations without auxiliary variables, because this more primitive language is more convenient for describing and reasoning about the system transformations we are interested in.

4 Parameter Abstraction

Assume $\mathcal{P}$ is a symmetric presentation as in Section 3. Fix a subset $M = \{p_1, \ldots, p_m\}$ of $N = \{p_1, \ldots, p_n\}$ and let $M_* = M + \{*\}$. Our goals in this section are: (1) to define the abstracted presentation $\mathcal{P}^m$ where the element $*$ serves as an abstraction of $p_{m+1}, \ldots, p_n$; (2) to define the abstraction map $\alpha\colon |\mathcal{P}| \to |\mathcal{P}^m|$; (3) to prove that α is a simulation. (One can see α as a complex example of "abstraction by state merging" [3].)

4.1 Syntactic Restrictions

Let us say that a type is *standard* if the parameter type N is not used in its construction. Define *array types* to be function types of the form $X \Rightarrow N$, $N \Rightarrow X$, or $N \Rightarrow N$ where X is a standard type. For arrays (that is, elements of array types) f, we will use the array indexing notation $f[x]$ instead of the function application $f(x)$. We will also write $f[x] := y$ as an abbreviation for the assignment (array update) $f := f'$ where f' agrees with f on all indices except x, where it has the value y. Finally, we define *admissible terms* as those in which, for any array variables f of type $N \Rightarrow X$, g of type $X \Rightarrow N$, and h of type $N \Rightarrow N$ one has that each occurrence of the variable must be part of an occurrence of a subterm $f[p_i]$, $g[e] = p_i$, or $h[p_i] = p_j$.

From now on, we require that the symmetric presentation $\mathcal{P}$ satisfies the following conditions:

- $p_1, \ldots, p_n$ and the equality operator on N are the only constants whose type is non-standard;
- the type of every system variable is either standard or is an array type; [5]
- in every rule, all subterms are admissible;
- in every rule, all assignments to array variables are of the form $f[p_i] := e$, $g[e'] := p_i$, and $h[p_i] := p_j$, where e and e' are terms of some standard type

[5] Variables of type N can be modelled as variables of array type $() \Rightarrow N$ with the unit index type $()$.

X, e' contains no occurrences of any p_i, and f, g, h have types $X \Rightarrow N$, $N \Rightarrow X$, $N \Rightarrow N$ respectively.

4.2 Definition of $\mathcal{P}^m$

Let $\mathsf{in}\colon M \to N$ be the inclusion and let $\mathsf{pr}\colon N \to M_*$ be the projection that restricts to the identity function on M and maps $N - M$ to $*$.

For each type T that may occur as a type of a system variable of $\mathcal{P}$, we define the corresponding *abstract type* $[T]$ and the *abstraction function* $\alpha_T : T \to [T]$ as follows:

$$\begin{aligned} [X] &= X & \alpha_X &= \mathsf{id}_X \\ [X \Rightarrow N] &= X \Rightarrow M_* & \alpha_{X\Rightarrow N} &= f \mapsto \mathsf{pr} \circ f \\ [N \Rightarrow X] &= M \Rightarrow X & \alpha_{N\Rightarrow X} &= f \mapsto f \circ \mathsf{in} \\ [N \Rightarrow N] &= M \Rightarrow M_* & \alpha_{N\Rightarrow N} &= f \mapsto \mathsf{pr} \circ f \circ \mathsf{in} \end{aligned}$$

We proceed to define the abstracted presentation $\mathcal{P}^m = (V, \iota_m, R_m)$. It uses the same variables as $\mathcal{P}$, but they are typed so that $\mathsf{type}(v) = [T]$ in $\mathcal{P}^m$ when $\mathsf{type}(v) = T$ in $\mathcal{P}$. Thus, with S and S_m denoting the state sets of $\mathcal{P}$ and $\mathcal{P}^m$, we have the abstraction map $\alpha\colon S \to S_m$ defined by $\alpha(s)(v) = \alpha_T(s(v))$, where $T = \mathsf{type}(v)$. The predicate ι_m and the rules R_m will be obtained by applying syntactic transformations A_{pred} and A_{rule} to ι and R respectively. These transformations, together with the corresponding transormations A_{term} and A_{asmt} of terms and assignments are defined in the following paragraphs.

For A_{term} there is no choice but to stipulate $A_{\text{term}}(v) = v$ for all $v \in V$, $A_{\text{term}}(p_i) = p_i$ for $i \leq m$, $A_{\text{term}}(p_i) = *$ for $i > m$, $A_{\text{term}}(w) = w$ for all other constants w, and then extend the definition inductively to all the terms: $A_{\text{term}}(e_1\ e_2) = (A_{\text{term}}(e_1))\ (A_{\text{term}}(e_2))$. (Recall that the only term-constructing operation is the function application.) By definition of A_{term},

$$A_{\text{term}}(e) = e \text{ if and only if } p_{m+1}, \ldots, p_n \text{ do not occur in } e. \tag{3}$$

Even though the syntactic expression $A_{\text{term}}(e)$ is well-defined for every term e, it may not be a legitimate term because of type mismatches, as in $e \equiv f[p_i]$, where $i > m$. However, the restriction of A_{term} to predicates can be turned into a total function A_{pred} as follows.

Suppose ϕ is a predicate and $\phi_1, \ldots, \phi_k$ are its subterms such that ϕ is a boolean combination of $\phi_1, \ldots, \phi_k$. Suppose also this boolean decomposition of ϕ is maximal in the sense that no ϕ_i is a boolean combination of its subpredicates. Clearly, $A_{\text{term}}(\phi)$ is the same boolean combination of $A_{\text{term}}(\phi_1), \ldots, A_{\text{term}}(\phi_k)$. Define $A_{\text{pred}}(\phi)$ to be the same boolean combination of terms $\phi'_1, \ldots, \phi'_k$, where

$$\phi'_i = \begin{cases} \phi_i & \text{if } A_{\text{term}}(\phi_i) = \phi_i \\ \mathsf{true} & \text{if } A_{\text{term}}(\phi_i) \neq \phi_i \text{ and } \phi_i \text{ occurs positively in } \phi \\ \mathsf{false} & \text{if } A_{\text{term}}(\phi_i) \neq \phi_i \text{ and } \phi_i \text{ occurs negatively in } \phi \end{cases}$$

It is easy to see that $A_{\text{pred}}(\phi)$ is well-typed for every ϕ.

All the properties we need of A_{pred} are stated in the following lemma. We should note that the lemma would hold for any function A_{pred} such that: (1) $A_{\text{pred}}(\phi) = \phi$ whenever $A_{\text{term}}(\phi) = \phi$, and (2) $\phi \Longrightarrow A_{\text{pred}}(\phi)$ is a tautology.

Lemma 4.1 *(a) If ϕ has no occurrences of constants $p_{m+1}, \ldots, p_n$, then $A_{pred}(\phi) = \phi$ and $\alpha(s) \models \phi$ is equivalent with $s \models \phi$.*
(b) If $s \models \phi$, then $\alpha(s) \models A_{pred}(\phi)$. □

The table below describes the assignment transformer A_{asmt}. The top row shows the four allowed forms for assignments (see Section 4.1); the bottom row shows the results of applying A_{asmt} to them. In the table, the symbol $\langle\rangle$ denotes the trivial assignment ("no assignment") and e' is either e or $\mathsf{undefined}$, depending on whether $A_{\text{term}}(e) = e$ or not.

$v := e$	$f[p_i] := e$	$g[e] := p_j$	$h[p_i] := p_j$
$v := e'$	$\langle\rangle$ if $i > m$ $f[p_i] := e'$ if $i \leq m$	$g[e] := \mathsf{pr}(p_j)$	$\langle\rangle$ if $i > m$ $h[p_i] := \mathsf{pr}(p_j)$ if $i \leq m$

Finally, we define the rule transformation function A_{rule} by

$$A_{\text{rule}}(\rho \rightarrowtriangle \{a_1 \ ; \ \ldots \ ; \ a_k\}) \ = \ A_{\text{pred}}(\rho) \rightarrowtriangle \{A_{\text{asmt}}(a_1) \ ; \ \ldots \ ; \ A_{\text{asmt}}(a_k)\},$$

then by setting $\iota_m = A_{\text{pred}}(\iota)$ and $\mathcal{R}_m = \{A_{\text{rule}}(r) \,|\, r \in R\}$ we complete the definition of $\mathcal{P}^m$.

The rules of $\mathcal{P}$ that contain no occurrences of $p_{m+1}, \ldots, p_n$ are unchanged by A_{rule} and so they are rules of $\mathcal{P}^m$ as well. The remaining rules of $\mathcal{P}^m$ will be called *degenerate*. Degenerate rules make the presentation $\mathcal{P}^m$ more complex than $\mathcal{P}$. For example, in the case when all rules of $\mathcal{P}$ are equivalent up to a permutation of N (that is, when $\mathcal{P}$ is equivalent to a parametrized presentation with only one quantified rule), a degenerate rule of $\mathcal{P}^m$ will not be equivalent modulo permutations of M to any non-degenerate rules, and may not be equivalent even to other degenerate rules.

4.3 Simulation

Theorem 4.2 *The abstraction map $\alpha\colon |\mathcal{P}| \to |\mathcal{P}^m|$ is a simulation.* □

Corollary 4.3 *If $\mathcal{P}^m \models \phi$ and $p_{m+1}, \ldots, p_n$ do not occur in ϕ then $\mathcal{P} \models \phi$.* □

For reference in Section 5, we need the following "lifting" lemma—a simple consequence of the fact that the subsystem of $\mathcal{P}^m$ defined by the set of non-degenerate rules is also a subsystem of $\mathcal{P}$.

Lemma 4.4 *If $\widetilde{s}_1 \to \ldots \to \widetilde{s}_k$ is an execution of $|\mathcal{P}^m|$ caused by non-degenerate rules, then for every s_1 such that $\alpha(s_1) = \widetilde{s}_1$ there exists an execution $s_1 \to \ldots \to s_k$ of $|\mathcal{P}|$ with $\alpha(s_i) = \widetilde{s}_i$ holding for every i.* □

5 The Method

The Chou-Mannava-Park method rests on the following theorem that combines guard strengthening and parameter abstraction.

Theorem 5.1 *Let $\mathcal{P}$ be a symmetric presentation with the parameter set $N = \{p_1, \ldots, p_n\}$ and that only the first m elements of N may occur in ϕ. Let $\mathcal{Q}$ be the presentation obtained from $\mathcal{P}^m$ by replacing each rule $A_{pred}(\rho) \rightarrowtriangle A_{asmt}(\boldsymbol{a})$ with $A_{pred}(\rho^\#) \rightarrowtriangle A_{asmt}(\boldsymbol{a})$, where the predicates $\rho^\#$ satisfy $\vdash \rho \wedge \phi_{sym} \Longrightarrow \rho^\#$. Then $\mathcal{Q} \models \phi$ implies $\mathcal{P} \models \phi$.* □

Now we can describe the Chou-Mannava-Park method abstractly and give a complete argument for its correctness.

The inputs are $\mathcal{P}$ and ϕ as in the theorem. First we construct $\mathcal{P}^m$ and then check $\mathcal{P}^m \models \phi$. If this happens to be true, we are done by Corollary 4.3 (which is the special case of Theorem 5.1 in which $\rho^\# = \rho$ for all guards ρ). Otherwise, the model-checker produces a counterexample execution

$$\widetilde{s}_0 \rightarrow \widetilde{s}_1 \rightarrow \ldots \rightarrow \widetilde{s}_k, \tag{4}$$

where $\widetilde{s}_0 \models \iota_m$ and $\widetilde{s}_k \not\models \phi$.

We claim that there must be a degenerate transition in (4) if $\mathcal{P} \models \phi$ holds. Indeed, assuming that every initial state of $\mathcal{P}^m$ lifts to an initial state of $\mathcal{P}$ (which will hold for non-pathological systems $\mathcal{P}$), if all transitions in (4) are non-degenerate, then by Lemmas 4.4 and 4.1(a), there is an execution $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_k$ of $\mathcal{P}$ with $s_0 \models \iota$ and $s_k \not\models \phi$, contradicting $\mathcal{P} \models \phi$.

We pick a degenerate transition $\widetilde{s}_i \rightarrow \widetilde{s}_{i+1}$ that we suspect is responsible for the spurious trace (4). Suppose this transition is based on a degenerate rule $A_{\text{pred}}(\rho) \rightarrowtriangle A_{\text{asmt}}(\boldsymbol{a})$ obtained by abstracting the rule $\rho \rightarrowtriangle \boldsymbol{a}$ of $\mathcal{P}$. Our guess that the predicate $A_{\text{pred}}(\rho)$ is too permissive and that this transition does not lift to $|\mathcal{P}|$ now needs to be supported by finding an invariant ("non-interference lemma") ψ_1 and a logical consequence $\rho^\#$ of $\rho \wedge \psi_1$ such that $\widetilde{s}_i \not\models A_{\text{pred}}(\rho^\#)$. Finding ψ_1 and $\rho^\#$ is the critical step that requires human intervention.

We create a new presentation $\mathcal{Q}_1$ by replacing the rule $A_{\text{pred}}(\rho) \rightarrowtriangle A_{\text{asmt}}(\boldsymbol{a})$ in $\mathcal{P}^m$ with $A_{\text{pred}}(\rho^\#) \rightarrowtriangle A_{\text{asmt}}(\boldsymbol{a})$. Then we check $\mathcal{Q}_1 \models \phi \wedge \psi_1$. If this happens to be true, we can conclude by Theorem 5.1 that ϕ (and ψ_1 as well) is an invariant of $\mathcal{P}$. Otherwise, the model-checker will produce a new counterexample. Analyzing this counterexample, just as above, we find a too permissive degenerate transition and find a putative invariant ψ_2 together with a new presentation $\mathcal{Q}_2$ that strengthens the guard of the rule of $\mathcal{Q}_1$ responsible for the offending transition. Then we check $\mathcal{Q}_2 \models \phi \wedge \psi_1 \wedge \psi_2$ etc. With luck, through a sequence of progressively finer approximations $\mathcal{P}^m, \mathcal{Q}_1, \mathcal{Q}_2, \ldots$ of $\mathcal{P}$ and non-interference lemmas $\psi_1, \psi_2, \ldots$ we will be able to prove $\mathcal{Q}_k \models \phi \wedge \psi_1 \wedge \cdots \wedge \psi_k$ for some k. As in the case of $\mathcal{Q}_1$, Theorem 5.1 will then imply that ϕ and the non-interference lemmas are invariants of $\mathcal{P}$.

References

[1] T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. Zuck. Parameterized verification with automatically computed inductive assertions. In *Proc. Conf. on Computer Aided Verification (CAV'01)*, volume 2102 of *LNCS*, pages 221–234. Springer, 2001.

[2] K. Baukus, Y. Lakhnech, and K. Stahl. Parameterized verification of a cache coherence protocol: Safety and liveness. In *Proc. Workshop on Verification, Model Checking, and Abstract Interpretation (VMCAI'02)*, volume 2294 of *LNCS*, pages 317–330, 2002.

[3] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, and P. McKenzie, editors. *Systems and Software Verification.* Springer, 2001.

[4] C.-T. Chou, P. K. Mannava, and S. Park. A simple method for parameterized verification of cache coherence protocols. In *Proc. Conf. on Formal Methods in Computer-Aided Design (FMCAD'04)*, volume 3312 of *LNCS*, pages 382–398. Springer, 2004.

[5] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proc. Conf. on Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 154–169. Springer, 2000.

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

[7] E. A. Emerson and V. Kahlon. Exact and efficient verification of parameterized cache coherence protocols. In *Proc. Conf. on Correct Hardware Design and Verification Methods (CHARME'03)*, volume 2860 of *LNCS*, pages 247–262. Springer, 2003.

[8] C. N. Ip and D. L. Dill. Better verification through symmetry. In *Proc. Conf. on Computer Hardware Description Languages and their Applications*, pages 97–111, 1993.

[9] S. K. Lahiri and R. E. Bryant. Constructing quantified invariants via predicate abstraction. In *Proc. Workshop on Verification, Model Checking, and Abstract Interpretation (VMCAI'04)*, volume 2937 of *LNCS*, pages 267–281, 2004.

[10] G. Lowe. On the application of counterexample-guided abstraction refinement and data independence to the parameterised model checking problem. In *Proc. Workshop on Automatic Verification of Infinite-State Systems (AVIS'04)*, ENTCS, 2004.

[11] K. L. McMillan. Parameterized verification of the FLASH cache coherence protocol by compositional model checking. In *Proc. Conf. on Correct Hardware Design and Verification Methods (CHARME '01)*, volume 2144 of *LNCS*, pages 179–195. Springer, 2001.

A Proofs

Proof of Proposition 2.1

Assuming $\mathcal{P}^{\#} \models \phi$, we prove by induction that for all reachable states s of $\mathcal{P}$ one has: $s \models \phi$ and s is $\mathcal{P}^{\#}$-reachable. The induction is on the length of the $\mathcal{P}$-execution that leads from an initial state to s. If s is $\mathcal{P}$-initial, then it is also $\mathcal{P}^{\#}$-initial, so there is nothing to prove.

Suppose then that $s \to s'$ is a $\mathcal{P}$-transition based on some rule $r \equiv \rho \twoheadrightarrow \boldsymbol{a}$, and suppose (the induction hypothesis) that $s \models \phi$ and s is $\mathcal{P}^{\#}$-reachable. Since $\mathcal{P}^{\#} \models \phi$, it suffices just to show that s' is $\mathcal{P}^{\#}$-reachable. Indeed, by our assumption, r fires at s, so we have $s \models \rho$ and it follows that $s \models \rho^{\#}$. Thus, the associated rule $r^{\#} \equiv \rho^{\#} \twoheadrightarrow \boldsymbol{a}$ fires at s (and leads to s'), so s' is $\mathcal{P}^{\#}$-reachable.

Proof of Lemma 4.1

(a) The first claim follows from (3). The second claim follows from the following observation, proved by induction on the structure of e.

$$\begin{array}{l} s(e) = \alpha(s)(e) \text{ holds for every state } s \text{ of } \mathcal{P} \text{ and every} \\ \text{admissible term } e \text{ without occurrences of } p_{m+1}, \ldots, p_n. \end{array} \tag{A.1}$$

(b) From definition of A_{pred}, it follows that $\phi \Longrightarrow A_{\text{pred}}(\phi)$ is a tautology. Thus, $s \models \phi$ implies $s \models A_{\text{pred}}(\phi)$, which in turn, by part (a) of the lemma, implies $\alpha(s) \models A_{\text{pred}}(\phi)$.

Proof of Theorem 4.2

By Lemma 4.1(b), if $s \models \iota$, then $\alpha(s) \models \iota_m$, so α maps initial states to initial states.

Suppose the rule $r = \rho \twoheadrightarrow \{a_1; \ldots; a_k\}$ causes a transition $s \to s'$. By Lemma 4.1(b), the rule $A_{\text{rule}}(r)$ fires at $\alpha(s)$, so it remains to check that the assignments of $A_{\text{rule}}(r)$ transform $\alpha(s)$ into $\alpha(s')$. Since the assignments $a_1, \ldots, a_k$ effect independent updates, there is no loss of generality in assuming $k = 1$. So suppose $r = \rho \twoheadrightarrow a$. There are four cases to consider, according to the table that defines A_{asmt} (Section 4). We give details for two of the cases and omit the other two to avoid repetition.

The first case to consider is when a is of the form $f[p_i] := e$, where f is an array variable of type $N \Rightarrow X$ in $\mathcal{P}$ and of type $M \Rightarrow X$ in $\mathcal{P}^m$. By definition, $\alpha(s)(f) = \alpha_{N \Rightarrow X}(s(f)) = s(f) \circ \mathsf{in}$, and similarly $\alpha(s')(f) = s'(f) \circ \mathsf{in}$. Thus,

- if $i > m$, then $\alpha(s')(f) = \alpha(s)(f)$;
- if $i \leq m$ and $A_{\text{term}}(e) = e$, then $\alpha(s')(f)[p_i] = (s'(f) \circ \mathsf{in})[p_i] = s'(f)[p_i] = s(e) = \alpha(s)(e)$, where the last equality follows from (A.1);
- $i \leq m$ and $A_{\text{term}}(e) \neq e$, then $A_{\text{rule}}(r) = (\rho \twoheadrightarrow f[p_i] := \mathsf{undefined})$.

In all three cases, we can conclude that $\alpha(s) \to \alpha(s')$ is a valid transition based on the rule $A_{\text{rule}}(r)$.

Consider now the case when a is of the form $g[e] := p_j$, where e is a term without occurrences of $p_1, \ldots, p_n$ and g is an array variable of type $X \Rightarrow N$ in $\mathcal{P}$ and of type $X \Rightarrow M_*$ in $\mathcal{P}^m$. We have

$$\alpha(s')(g)[\alpha(s)(e)] = \alpha(s')(g)[s(e)] = \alpha_{X\Rightarrow N}(s'(g))[s(e)] = \mathsf{pr}(s'(g)[s(e)]) = \mathsf{pr}(p_j),$$

where the first equality follows from (A.1), the second from definition of the abstraction map $\alpha\colon |\mathcal{P}| \to |\mathcal{P}^m|$, the third from the definition of $\alpha_{X\Rightarrow N}$, and the fourth from the fact that the assignment $g[e] := p_j$ transforms s into s'.

Proof of Corollary 4.3

By Theorem 4.2, α maps reachable states of $\mathcal{P}$ to reachable states of $\mathcal{P}^m$. By Lemma 4.1, ϕ holds in s when it holds in $\alpha(s)$. Thus, if ϕ holds in all reachable states of $\mathcal{P}^m$, it must holds in all reachable states of $\mathcal{P}$.

Proof of Theorem 5.1

Note first that Theorem 5.1 cannot be quickly proved from "$\mathcal{Q} \models \phi$ implies $\mathcal{P}^m \models \phi$ by Proposition 2.1" and "$\mathcal{P}^m \models \phi$ implies $\mathcal{P} \models \phi$ by Corollary 4.3" because the first of these claims is false. To apply Proposition 2.1, we would need $\vdash A_{\text{pred}}(\rho) \wedge \phi \Longrightarrow A_{\text{pred}}(\rho^{\#})$, which is not necessarily true.

We define an auxiliary symmetric system presentation $\mathcal{P}'$ and the corresponding parameter abstraction $\mathcal{P}'_m$. The proof of the theorem will follow from the implication chain

$$\mathcal{Q} \models \phi \quad \overset{(a)}{\Longrightarrow} \quad \mathcal{P}'_m \models \phi \quad \overset{(b)}{\Longrightarrow} \quad \mathcal{P}' \models \phi \quad \overset{(c)}{\Longrightarrow} \quad \mathcal{P} \models \phi. \tag{A.2}$$

To define $\mathcal{P}'$, we need to introduce some notation. For each rule r of $\mathcal{P}$, let ρ_r and $\boldsymbol{a}_r$ denote the guard and the assignment part of r respectively. (In other words, $r \equiv \rho_r \rightarrowtriangle \boldsymbol{a}_r$.) Note that the rules of $\mathcal{Q}$ are of the form $A_{\text{pred}}(\rho_r^{\#}) \rightarrowtriangle A_{\text{asmt}}(\boldsymbol{a}_r)$, where r is a rule of $\mathcal{P}$.

Define predicates $\rho_r^* = \bigwedge_\pi \pi^{-1}(\rho^{\#}_{\pi(r)})$, where the conjunction ranges over all permutations π of N. Finally, define $\mathcal{P}'$ to be the presentation given by the rules $\rho_r^* \rightarrowtriangle \boldsymbol{a}_r$, where r is a rule of $\mathcal{P}$. It is easy to check that $\mathcal{P}'$ is a symmetric presentation. [6]

Now we can prove the implications in (A.2). For (a), note that the rules of $\mathcal{P}'_m$ are of the form $A_{\text{pred}}(\rho_r^*) \rightarrowtriangle A_{\text{asmt}}(\boldsymbol{a}_r)$ and that for each such rule there is a corresponding rule $A_{\text{pred}}(\rho_r^{\#}) \rightarrowtriangle A_{\text{asmt}}(\boldsymbol{a}_r)$ of $\mathcal{Q}$. Since $\vdash \rho_r^* \Longrightarrow \rho_r^{\#}$ and since A_{pred} is monotonic with respect to implication, it follows that every rule of $\mathcal{P}'_m$ is less permissive than the corresponding rule of $\mathcal{Q}$. This proves (a).

[6] Note, however, that the set of rules $\rho_r^{\#} \rightarrowtriangle \boldsymbol{a}_r$ is not necessarily symmetric because we have not made any assumptions on the equivariance of the map $\rho_r \mapsto \rho_r^{\#}$.

The implication (b) is an instance of Corollary 4.3. For the implication (c), note first that the assumption $\mathcal{P}' \models \phi$ can be strengthened to $\mathcal{P}' \models \phi_{sym}$ (Lemma 3.1). By Proposition 2.1, we can derive $\mathcal{P} \models \phi_{sym}$ (and $\mathcal{P} \models \phi$ with it) if we can only prove $\phi_{sym} \wedge \rho_r \vdash \rho_r^*$ holds for all r. By definition of ρ_r^*, it suffices to prove $\vdash \phi_{sym} \wedge \rho_r \Longrightarrow \pi^{-1}(\rho^{\#}_{\pi(r)})$ for every r and every permutation π. This is equivalent to $\pi(\phi_{sym} \wedge \rho_r) \vdash \rho^{\#}_{\pi(r)}$. We have $\vdash \pi(\phi_{sym}) \Longleftrightarrow \phi_{sym}$ since ϕ_{sym} is symmetric, and also $\vdash \pi(\rho_r) \Longleftrightarrow \rho_{\pi(r)}$ since the presentation $\mathcal{P}$ is symmetric. Therefore, our last goal can be restated as $\vdash \phi_{sym} \wedge \rho_{\pi(r)} \Longrightarrow \rho^{\#}_{\pi(r)}$, which is an assumption in the statement of the theorem.