Symmetry and Completeness in the Analysis of Parameterized Systems

Kedar S. Namjoshi

Bell Labs
kedar@research.bell-labs.com

Abstract. It is shown that the *cutoff* method—which summarizes a parameterized system by a finite set of its instances—is complete for proving safety properties. This implies completeness of other, less stringent, proof methods for parameterized verification. It is shown that the cutoff method is equivalent to determining a (parameterized) inductive invariant. The second part of the paper describes a new algorithm to construct universally quantified, parameterized inductive invariants. This algorithm is shown to compute the *strongest* invariant of a given shape, and is complete under certain conditions. A key observation is a previously unnoticed connection between inductiveness, small model theorems, and compositional analysis.

1 Introduction

Parameterization is ubiquitous in programming: most programs are parameterized in some manner (e.g., the size of buffers, or the number of threads). A particularly fascinating case is that of network protocols which are composed of a variable number of finite-state, isomorphic processes. Any particular instance of this parameterized system is finite-state, and its correctness can be determined automatically through model checking. But the real goal is to verify *all*—i.e., an infinite number of—instances.

In this regard, a common experience is that the correctness argument for small instances includes all the case analysis required for full correctness. It is tempting to conjecture that there is a small bound such that the correctness of instances up to that bound suffices to establish correctness in general. Unfortunately, this conjecture is incorrect. Apt and Kozen [2] showed that the parameterized verification question—showing correctness of all instances—is undecidable, even if the component processes are finite-state and isomorphic (cf. [42]). This negative result has led naturally to two forms of analyses: (i) showing decidability for restricted classes of protocols (cf. [21,17,15]), and (ii) generally applicable, semi-automated proof principles based on induction and abstraction.

This article focuses on the general, type (ii) methods, which exploit the symmetry of the problem to reduce or to simplify proof obligations. Let P^n represent the parameterized system $P_0||P_1||...||P_{n-1}$, produced by asynchronous composition of n isomorphic copies of a process P. A central concept is that the behavior

B. Cook and A. Podelski (Eds.): VMCAI 2007, LNCS 4349, pp. 299–313, 2007.
© Springer-Verlag Berlin Heidelberg 2007

of arbitrary instances of a protocol can be summarized by a finite-state process. The *closure process*, introduced in [9,7], is precisely such a summary. The *process invariant* method, introduced in [32,43], requires that the summarizing process, I, is also invariant, in the sense that the behavior of both P and $P||I$ is simulated by I — the stronger requirement makes it easier to show that I is a closure. The *cutoff method*, implicit in [32,21] and made explicit in [17], goes further and requires that the process invariant be a union of instances up to a (typically small) cutoff bound K. (i.e., $I = P^1 + P^2 + \ldots + P^K$). The cutoff method formalizes the verifier's intuition that all interesting patterns of behavior are already present in instances of a small size.

These proof methods have been applied successfully to several protocols, but it is not immediately apparent whether they are universally applicable. Does every correct protocol have a cutoff proof? If so, is the cutoff proportional to the individual process size?

1.1 Contributions

The first part of the paper analyzes completeness for safety (invariance) properties. It shows that an invariance property (e.g., "the protocol ensures mutual exclusion") which holds of the protocol can always be shown using a cutoff of 1 — I.e., by showing that the smallest instance simulates instances of arbitrary size, and is itself correct. This implies completeness for the two other methods. Known bounds for cutoffs[1] are either property-dependent (cf. [32,17,15]), or are exponentially large in the process description [21,16]. The cutoff of 1 holds also for arbitrary linear-time properties, and for multi-parameter systems. These results provide formal justification to the intuition regarding small instances. The proof also shows that a cutoff-based proof is equivalent to the construction of an *inductive* parameterized invariant.

The second part of the paper, therefore, focuses on automatic methods of computing inductive parameterized invariants. The starting point is a method of "invisible invariants", proposed by Pnueli, Ruah, and Zuck in [39]. The central idea is to generalize from the reachable states of small instances into an assertion of the shape $(\forall i : \varphi(i))$. Remarkably, this simple heuristic suffices to construct inductive assertion proofs of safety for several protocols. On the other hand, it is known to be incomplete: for some protocols, it fails to construct a quantified inductive invariant, even though one is known to exist.

This paper demonstrates a previously unnoticed connection between the inductiveness of quantified assertions and compositional reasoning. The key observation is that the *only* assertions of the shape $(\forall i : \varphi(i))$ which are inductive are those where actions of process $P(j)$ do not "interfere" with the inductiveness of $\varphi(i)$. This connection gives rise to an algorithm, called the *"split-invariant method"*, for the generation of quantified inductive invariants. The algorithm constructs the strongest assertion φ (in a restricted logic) such that $(\forall i : \varphi(i))$

[1] These are cutoffs for decidability, so they apply also to showing that a protocol incorrect, whereas the cutoff of 1 applies only to a correctness proof.

is inductive for all instances. This fact, combined with a small model theorem from [39], ensures completeness. As pleasing side-effects, non-interference explains the incompleteness of the inductive invariant method (reachable states do not always form non-interfering assertions), and also explains the occasional need to introduce auxiliary variables—it is known from work by Owicki and Gries [38] and Lamport [35] that this is necessary for non-interference. The method has been implemented with TLV [40]. Initial results show that completeness is not achieved at the cost of efficiency.

2 Completeness of the Cutoff Method

Programs are given meaning through transition systems. A transition system over a set of atomic propositions AP is a tuple (I, S, R, L), where S is a set of states, I is a subset of S, the initial states, R is a subset of $S \times S$, the transition relation, and $L : S \times 2^{AP}$ is a labeling function that assigns a subset of propositions to each state. If (s, s') is a tuple in R, s' is a *successor* of s, and there is a *transition* from s to s'. A state s is *reachable* if there is a finite sequence $s_0, s_1, \ldots s_k$ where s_0 is in I, s_{i+1} (where defined) is a successor of s_i for each i, and $s_k = s$. Transition systems are assumed to be *left-total*, i.e., every state has a successor. A *state predicate* is a Boolean combination of atomic propositions. The satisfaction of a predicate p at a state, written $s \models p$ or $p(s)$, is defined in the usual way by induction on formula structure. A state predicate φ is *invariant* of M if it holds at all reachable states of M.

Definition 0. *For transition systems M and N, and a set of state predicates SP, a relation $X : X \subseteq S_M \times S_N$ is a* simulation respecting SP *if:*

- *(initiality) for every initial state s of M, there is an initial state t of N such that (s, t) is in X,*
- *(step) for every (s, t) in X, and every successor s' of s, there is a successor t' of t such that (s', t') is in X,*
- *(label) for every pair (s, t) in X, and every predicate p in SP, $p_M(s) \equiv p_N(t)$*

Definition 1. *For transition systems M and N, and state predicates SP, a relation $X : S_M \times S_N$ is a* bisimulation respecting SP *if both X and its inverse are simulations respecting SP.*

Two transition systems are "(bi)simular *upto* SP" if there exists a (bi)simulation relation respecting SP between the transition systems. The usual definition of bisimulation between Kripke structures can be recovered by setting $SP = AP$.

Inductiveness, Invariance, and Fixpoints. A state assertion ξ is *inductive* for M if it is implied by the initial condition for M and it is preserved by every transition of M. This is formalized by the conditions (1) (initiality) and (2) (inductiveness) below. Here, *wlp* is the weakest liberal precondition transformer

introduced by Dijkstra [13]. The notation $[\psi]$, from Dijkstra and Scholten [14], indicates that ψ is valid.

$$[I_M \;\Rightarrow\; \xi] \tag{1}$$
$$[\xi \;\Rightarrow\; wlp(M,\xi)] \tag{2}$$

An inductive assertion is *adequate* to show the invariance of a state property φ if it implies φ (condition (3)).

$$[\xi \;\Rightarrow\; \varphi] \tag{3}$$

Applying the duality (the Galois connection) between *wlp* and the strongest post-condition operator, *sp*, condition (2) is equivalent to

$$[sp(M,\xi) \;\Rightarrow\; \xi] \tag{4}$$

Theorem 0. *(Knaster-Tarski) A monotonic function f on a complete lattice has a least fixpoint, which is also the strongest solution to $X : [f(X) \;\Rightarrow\; X]$, and a greatest fixpoint, which is the weakest solution to $X : [X \;\Rightarrow\; f(X)]$.*

For *finite* lattices, the least fixpoint can be computed as the limit of the sequence $X_0 = \bot; X_{i+1} = f(X_i)$. The greatest fixpoint can be computed as the limit of the sequence $X_0 = \top; X_{i+1} = f(X_i)$, where $\top$ and $\bot$ are, respectively, the top and bottom elements of the lattice.

Taken together, conditions (1) and (4) give $[(I_M \vee sp(M,\xi)) \;\Rightarrow\; \xi]$. As function $f(\xi) = I_M \vee sp(M,\xi)$ is monotonic, by the Knaster-Tarski theorem, it has a least fixpoint, which is the set of reachable states of M. Taken together, conditions (2) and (3) give $[\xi \;\Rightarrow\; (\varphi \wedge wlp(M,\xi))]$. The function $g(\xi) = \varphi \wedge wlp(M,\xi)$ is monotonic, hence there is a greatest solution in ξ. This is expressed in CTL as $\mathsf{AG}(\varphi)$. If φ is invariant for M, the reachable states of M additionally satisfies (3), and $\mathsf{AG}(\varphi)$ additionally satisfies (1).

2.1 Completeness

Lemma 0. *Any pair of transition systems M, N which satisfy the invariance property $\mathsf{AG}(\varphi)$ are bisimular upto $\{\varphi\}$.*

Proof. Define the relation X between states of M and N by $(s,t) \in X$ iff $M, s \models \mathsf{AG}(\varphi)$ *and* $N, t \models \mathsf{AG}(\varphi)$. We have to show that X and its inverse are simulation relations respecting $\{\varphi\}$.

(initiality) consider initial states s and t of M and N respectively. By the assumption, both s and t satisfy $\mathsf{AG}(\varphi)$, and are therefore related by X.

(step) Let s, t be such that $(s,t) \in X$, and let s' be a successor of s. As s satisfies $\mathsf{AG}(\varphi)$, by the fixpoint formulation, all of its successors also satisfy $\mathsf{AG}(\varphi)$. For the same reason, all successors of t satisfy $\mathsf{AG}(\varphi)$, and t has at least one successor, say t', as N is left-total. Thus, (s', t') is a pair in X. A symmetric argument establishes the step property for the inverse of X.

(label) Let s, t be such that $(s,t) \in X$. As s and t satisfy $\mathsf{AG}(\varphi)$, by the fixpoint formulation, they satisfy φ. As φ is a state predicate, $\varphi_M(s) \equiv \varphi_N(t)$. □

The cutoff method is defined below for arbitrary parameterized programs—not just those arising from a composition of processes. A parameterized program is a family of transition systems $\{M(n)\}$, indexed by a parameter n. The property is also parameterized by n; for instance, mutual exclusion between critical sections, identified by C, can be specified by $\varphi(n) = (\forall i, j : i, j \in [0..n-1] \wedge i \neq j : \neg(C_i \wedge C_j))$.

Definition 2. *[**Cutoff Method**] To show that a parameterized program $M(n)$ satisfies a similarly parameterized invariance property $\mathsf{AG}(\varphi(n))$ for all n, find a cutoff bound K such that:*

- *For every $n : n > K$, there is $j : j \leq K$ such that $M(n)$ is simulated upto φ by $M(j)$, and*
- *For every $j : j \leq K$, $M(j)$ satisfies $\mathsf{AG}(\varphi(j))$.*

Theorem 1. *The cutoff method is complete for invariance properties: i.e., if $M(n)$ satisfies $\mathsf{AG}(\varphi(n))$ for all n, there is a cutoff bound K such that the conditions are met. In particular, $K = 1$ suffices.*

Proof. Let $K = 1$. As $M(n)$ satisfies $\mathsf{AG}(\varphi(n))$ for all n, the second condition is met: $M(1)$ satisfies $\mathsf{AG}(\varphi(1))$.

Consider $M(n)$, for any $n : n > 1$. As $M(n)$ satisfies $\mathsf{AG}(\varphi(n))$, and $M(1)$ satisfies $\mathsf{AG}(\varphi(1))$, a proof on the lines of that for Lemma 0 shows that $M(1)$ and $M(n)$ are bisimular upto φ. The only modification to the previous proof is in the definition of X. A pair (s,t) is in X (where s is a state of $M(1)$ and t is a state of $M(n)$) if $M(1), s$ satisfies $\mathsf{AG}(\varphi(1))$ and $M(n), t$ satisfies $\mathsf{AG}(\varphi(n))$. This proves the first condition. □

To illustrate the workings of this theorem by an example, consider the simple mutual exclusion protocol[2] described in Figure 1. The desired invariant predicate $\varphi(n)$ is $(\forall i, j : i, j \in [n] \wedge i \neq j : \neg(C_i \wedge C_j))$, where C_i is shorthand for $(st(i) = C)$. It is easily checked that this predicate is **not** inductive. The completeness proof works with any predicate that is inductive and is stronger than φ. One such predicate is $\xi(n) = (\forall i, j : i, j \in [n] \wedge i \neq j : (C_i \vee E_i) \Rightarrow (x \wedge \neg(C_j \vee E_j)))$. The bisimulation relation defined in the completeness proof relates state s of P^n with t of P^1 if they agree on ξ. As $\xi(1) = \mathit{true}$, every state of P^n is bisimular (in the new sense) to any state of P^1!

The completeness proof constructs a bisimulation from $\mathsf{AG}(\varphi(n))$. In fact, any inductive invariant stronger than φ will do: $\mathsf{AG}(\varphi)$ is simply the weakest such assertion. Determining an appropriate simulation relation, and finding a proof of a candidate relation, are both difficult questions, which are undecidable in general. Similarly, determining whether a protocol has a particular cutoff bound is also undecidable, in general. The following theorem establishes the converse.

[2] This protocol is taken from [39] but with a change of notation.

```
var x: boolean /* semaphore */
initially x=true

process P(i) ::
  var st: {I,T,C,E}
  initially st=I
  do
  | (st=I) -> st := T
  | (st=T) and x -> st,x := C,false
  | (st=C) -> st := E
  | (st=E) -> st,x := I,true
  od
```

Fig. 1. Mutual Exclusion Protocol

Theorem 2. *A cutoff proof of invariance for program $M(n)$ and property $\varphi(n)$ induces an assertion $\xi(n)$ that is inductive for $M(n)$, and implies $\varphi(n)$.*

3 Parameterized Invariants and Non-interference

The discussion in Section 2 showed that the existence of cutoffs for invariance properties is equivalent to the existence of parameterized inductive invariants. The "invisible invariant" method of [39] computes inductive predicates of the shape $(\forall i : \theta(i))$, where θ is a limited assertion. It consists of two steps: in the first step, a candidate for θ is constructed by generalizing the set of reachable states of small instances up to a cutoff size N_0; in the second step, the generated candidate, $(\forall i : \theta(i))$, is checked for inductiveness and adequacy for arbitrary N. The second step can be automated based on a *small model theorem*, which shows that these properties need to be checked only up to a second cutoff N_1.

In a series of papers [3,4] the authors showed that this simple heuristic suffices to show correctness of a number of parameterized protocols. On the other hand, the authors also point out that this method is not guaranteed to succeed—even if a suitable invariant exists, generalization from the reachable states may fail for some protocols [4].

To understand the source of incompleteness, it is helpful to reason in reverse from the goal: how does an assumption of inductive invariance for $(\forall i : \theta(i))$ constrain θ? A first consequence is that $\theta(0)$ must be inductive for process 0 in an instance of size 1. Thus, it suffices to enumerate every inductive invariant for process 0 in $M(1)$, and check inductiveness for all N by applying the small model theorem. (The set of inductive assertions forms a lattice with top element $\mathsf{AG}(\varphi(1))$ and bottom element the reachable states.) While complete, the procedure is extremely inefficient, as there could be exponentially many inductive invariants for process 0. (If a process state is described by k Boolean variables, there are 2^{2^k} distinct Boolean expressions to consider.) While this line of reasoning does not lead to a useful algorithm, it does suggest that generalizing the

set of reachable states — just one out of many candidates — need not give rise to a parameterized invariant.

Let us continue by analyzing the case $N = 2$. Parameterized inductiveness of $(\forall i : \theta(i))$ implies the following inductiveness conditions for $N = 2$. (The notation $P_0(2)$ represents process 0 in a 2-process composition $M(2) = P_0||P_1$. The symmetric condition for $P_1(2)$ is not shown.)

$$[I_{M(2)} \Rightarrow (\theta(0) \wedge \theta(1))] \tag{5}$$

$$[(\theta(0) \wedge \theta(1)) \Rightarrow wlp(P_0(2), \theta(0) \wedge \theta(1))] \tag{6}$$

As *wlp* distributes over conjunction, (6) is equivalent to

$$[(\theta(0) \wedge \theta(1)) \Rightarrow wlp(P_0(2), \theta(0))] \tag{7}$$

$$[(\theta(0) \wedge \theta(1)) \Rightarrow wlp(P_0(2), \theta(1))] \tag{8}$$

The first is expected: it says (roughly) that $\theta(0)$ is preserved by transitions of process P_0. The second is quite different in nature: it says that $\theta(1)$ is *also* preserved by transitions of process P_0—i.e., process P_0 *does not interfere* with the invariance of $\theta(1)$. The notion of non-interference was introduced by Owicki and Gries in their seminal work on compositional reasoning [38]. The rest of this section explores this connection in detail.

3.1 Background

The class of parameterized programs to which the invisible invariant method is applied is called *bounded-data discrete systems* (BDS's) [39,3]. Program variables can be Boolean and finite-domain variables (type $\mathbf{T}_0$), variables with ranges $[0..N_i - 1]$, for a parameter N_i (types $\mathbf{T}_i$), and arrays with domain $\mathbf{T}_i$ and elements from $\mathbf{T}_j$ for some i, j (types $\mathbf{T}_i \mapsto \mathbf{T}_j$).

Atomic formulas can only compare for inequality basic expressions of the same type. Basic expressions are limited to variables x, or 1-level indexed entries $Z[x]$ (not $Z[Z[x]]$). Formulas are built up from atomic formulas by Boolean combination, and quantification. The transition relation of a parameterized system is constrained to the shape $(\exists h_1, h_2, \ldots, h_k : (\forall t_1, t_2, \ldots, t_m : R(h, t)))$, where h_i, t_i are vectors of variables of $\mathbf{T}_i$, for all i. Intuitively, the h-variables identify the process that makes a transition, the t-variables are used to express the constraint that a transition by one process leaves the local state of all other processes unchanged. The expression defining the initial condition is assumed to be symmetric (i.e., left unchanged by permutations of indices). For example, the protocol from Figure 1 has types $x : \mathbf{T}_0$ and $state : \mathbf{T}_1 \mapsto \mathbf{T}_0$; its initial condition is $x \wedge (\forall i : state(i) = I)$.

Candidate invariance assertions have the shape $(\forall i_1, i_2, \ldots, i_k : \theta(i_1, i_2, \ldots, i_k))$, where i_m is a vector of variables of type $\mathbf{T}_m$, for each m, and the variables of any given type are given distinct values. An example is the mutual exclusion property: $(\forall i, j : i \neq j : \neg(C_i \wedge C_j))$, where $i, j : \mathbf{T}_1$.

3.2 Non-interference and Split Invariants

Consider a program, $A||B$, formed by the *asynchronous* composition of processes A and B, and a correctness property φ. Let A (B) have state defined by sets of variables V_A (V_B). A *local* assertion (for A; symmetrically for B) is an assertion formed from V_A. Variables local to A are denoted by $L_A = V_A \setminus V_B$, while the global (shared) variables are those in $V_A \cap V_B$.

A pair of assertions (θ_A, θ_B), which are local over A and B, respectively, is called a *split assertion*. A split assertion (θ_A, θ_B) is a *split invariant* if $\theta_A \wedge \theta_B$ is an inductive invariant for $A||B$. For asynchronous composition, the invariance conditions obtained from (1)-(3) simplify to the following.

$$[I_{A||B} \;\Rightarrow\; (\theta_A \wedge \theta_B)] \tag{9}$$

$$[(\theta_A \wedge \theta_B) \;\Rightarrow\; wlp(A, (\theta_A \wedge \theta_B))] \tag{10}$$

$$[(\theta_A \wedge \theta_B) \;\Rightarrow\; wlp(B, (\theta_A \wedge \theta_B))] \tag{11}$$

A split invariant satisfies a property φ if

$$[(\theta_A \wedge \theta_B) \;\Rightarrow\; \varphi] \tag{12}$$

A k-split assertion over k processes $P_0||P_1 \ldots ||P_{k-1}$ (these are not necessarily isomorphic) is a k-vector of the form $(\theta_0, \ldots, \theta_{k-1})$ where each θ_i is defined over the variables of P_i. It is a k-split invariant if $(\forall i : \theta_i)$ is an inductive invariant of the k-process system.

Computing the strongest split invariant. Applying the Galois connection between *wlp* and *sp* to (10) and (11), combining with (9), and re-arranging results in the following equivalent formulation. The existential quantification encodes locality: e.g., variables quantified out in (13) are irrelevant to θ_A by locality.

$$[(\exists L_B : sp(A, \theta_A \wedge \theta_B) \vee sp(B, \theta_A \wedge \theta_B) \vee I) \;\Rightarrow\; \theta_A] \tag{13}$$

$$[(\exists L_A : sp(A, \theta_A \wedge \theta_B) \vee sp(B, \theta_A \wedge \theta_B) \vee I) \;\Rightarrow\; \theta_B] \tag{14}$$

Implications (13) and (14), in turn, can be written as the pre-fixpoint formulation: $[\mathcal{F}(\theta_A, \theta_B) \preceq (\theta_A, \theta_B)]$, where $\mathcal{F}$ is the pair function formed by the left-hand expressions in the implication, and $\preceq$ denotes pair-wise implication. Since $\mathcal{F}$ is monotone over (θ_A, θ_B) according to $\preceq$ (*sp* is a monotone function), by the Knaster-Tarski theorem, $\mathcal{F}$ has a *least* fixpoint, which, by construction, is also the least solution to conditions (9)-(11).

The Knaster-Tarski algorithm applied to $\mathcal{F}$ results in the least fixpoint (θ_A^*, θ_B^*), which is the *strongest* inductive split assertion. The operations required to evaluate $\mathcal{F}$ (the computation of *sp*, existential quantification) can be carried out with BDD's for finite variable domains.

Theorem 3. *(Completeness of the procedure) There exists an inductive split-invariant that is adequate for φ if, and only if, $[(\theta_A^* \wedge \theta_B^*) \;\Rightarrow\; \varphi]$.*

This method is easily extended to compute the strongest k-split invariant over a k-process instance. The general form of $\mathcal{F}$ is represented as $\mathcal{F}^k$, which is a vector of functions over a k-split assertion $\theta = (\theta_0, \ldots, \theta_{k-1})$. For each $i \in [k]$, the ith component of $\mathcal{F}^k(\theta)$ is given below, where $L = (\cup i : L_i)$ is the set of local variables.

$$\mathcal{F}_i^k(\theta) = (\exists L \setminus L_i : init(k) \vee (\exists j : sp(P_j(k), (\forall m : \theta_m)))) \tag{15}$$

For a parameterized system, the symmetry in its definition ensures that, during the fixpoint calculation, it is necessary only to compute the new value for a single component, say $\theta(0)$; $\theta(i)$ is constructed by applying the substitution $0 \mapsto i$.

Theorem 4. *(Symmetric split invariant) For a parameterized instance, P^k, with a symmetric initial condition, there is an assertion $\theta^*(V_i)$ such that the least fixpoint of $\mathcal{F}^k$ is given by $(\forall i : \theta^*(V_i))$.*

3.3 Quantified Inductive Invariants

The previous discussion showed that, for a parameterized system, (i) for any *inductive* quantified assertion $(\forall i : \theta(i))$, the vector $(\theta(0), \ldots, \theta(k-1))$ is a symmetric k-split invariant for all k; and (ii) for any k, the *strongest* k-split invariant can be computed through a least fixpoint procedure, and this is symmetric. In this section, we close the loop by showing that, given a small model property for an assertion logic $\mathcal{L}$ with bound K, the assertion $(\forall i : \theta^*(i))$ induced by the strongest K-split invariant is an inductive parameterized invariant.

Definition 3. *("Near-inductive" invariant) A quantified assertion (e.g., $(\forall i : \theta(i))$) is* K-inductive *if it is inductive for all instances of size at least K. An assertion is* near-inductive *if it is K-inductive for some K.*

Definition 4. *(Small Model Property) For any θ in $\mathcal{L}$, checks of the form (9)-(12) for an assertion $(\forall i : \theta(i))$ are valid for all N if, and only if, there exists K such that they are valid for all instances of size up to K. The bound K may depend on the quantification over i, but should be independent of θ.*

A *strong* small model property holds if validity need be checked only for the instance of size K. In [3], the small model property is shown for BDS's with variables of type $\mathbf{T}_0, \mathbf{T}_1$, and $\mathbf{T}_1 \mapsto \mathbf{T}_2$. The proof can be strengthened to show the stronger form of this property.

Let V_i be the set of variables accessed by process i (local as well as global). The computation of K-split invariants on general BDS's does not necessarily produce assertions that are expressible in the logic $\mathcal{L}$. In order to ensure this, let α_i be the closure operator, defined as follows: $\alpha_i^K(S)$, for an index $i \in [K]$ and a set of states S in $M(K)$, is the smallest superset of S that is expressible in the logic $\mathcal{L}$ together with the assertion $y = i$ for each $\mathbf{T}_1$ variable y (this is

well-defined if $\mathcal{L}$ is closed under arbitrary intersection). The closure operator can be computed using BDD's if $\mathcal{L}$ is based on a finite set of predicates[3].

The condition $[\alpha_i^K(\theta(V_i)) \Rightarrow \theta(V_i)]$ is added to the fixpoint formulation for $\mathcal{F}$. This defines a new monotonic operator, written $\mathcal{F}_{\alpha,i}^K(\theta) = \alpha_i^K(\theta(V_i)) \vee \mathcal{F}_i^K(\theta)$. Not only is any pre-fixpoint $\theta(V_i)$ of this operator a pre-fixpoint of $\mathcal{F}_i^K$ (which is desired for inductiveness), it is also a pre-fixpoint of α_i^K, which ensures that (as α is a closure) $[\alpha_i^K(\theta(V_i)) \equiv \theta(V_i)]$; i.e., $\theta(V_i)$ is expressible in $\mathcal{L}$.

Theorem 5. *(Completeness) For a BDS with a strong small model property and bound K, and assertions of the type $(\forall i : \theta(V_i))$, let $\theta^*(V_i)$ describe the symmetric K-split invariant computed as the strongest fixpoint of $\mathcal{F}_\alpha^K$. Then, $(\forall i : \theta^*(V_i))$ is K-inductive, and this is the strongest such assertion.*

Proof. By the preceding discussion, any fixpoint of $\mathcal{F}_\alpha^K$ is expressible in the logic $\mathcal{L}$ and is a K-split invariant.

We need to show K-inductiveness, i.e., that $(\forall i : \theta^*(V_i))$ is inductive for all $N : N \geq K$. The proof is by contradiction. If the assertion is non-inductive for some N, the strong form of the small model property implies that it is non-inductive for the instance of size K, which contradicts the assumption that it is a K-split invariant. Note that it is important that the bound K is independent of the particular θ^*.

Any K-inductive assertion of the shape $(\forall i : \theta(V_i))$ that is expressible in $\mathcal{L}$ is a K-split invariant. This implies that θ is a pre-fixpoint of $\mathcal{F}_\alpha^K$. As θ^* is the strongest pre-fixpoint, $[\theta^*(V_i) \Rightarrow \theta(V_i)]$; hence, $[(\forall i : \theta^*(V_i)) \Rightarrow (\forall i : \theta(V_i))]$. □

The strong form of the small model property can be dispensed with, at the cost of computing a weaker inductive assertion. For a simpler notation, this is explained for $\mathcal{F}^K$, not $\mathcal{F}_\alpha^K$. The computation of θ^* is set up so that it is a k-split invariant, for *every* $k : k \leq K$. Recall, from the discussion following Theorem 4, that the computation of a k-split invariant can be carried out by computing $\theta(V_0)$ using $\mathcal{F}_0^k(\theta)$, deriving the others by substitution. Thus, consider the assertion that $[(\mathcal{F}_0^k(\theta))(V_0) \Rightarrow \theta(V_0)]$ for all $k : k \leq K$. This is equivalent to

$$[(\exists k : k \leq K : (\mathcal{F}_0^k(\theta))(V_0)) \Rightarrow \theta(V_0)] \tag{16}$$

The left-hand side of the implication defines a new monotonic operator: $\mathcal{G}_0^K(\theta) = (\exists k : k \leq K : \mathcal{F}_0^k(\theta))$. This is used in place of $\mathcal{F}_\alpha^K$ in Theorem 5. While this substitution ensures inductiveness for all N, the additional $\mathcal{F}$ computations that are required could cause computational difficulty in practice. Moreover, as $\mathcal{G}^K$ is weaker than $\mathcal{F}^K$, the fixpoint that is produced is also a weaker assertion. This makes it less likely to satisfy the correctness condition. From these considerations, it appears that the weaker K-inductiveness requirement is preferable to inductiveness for all N.

[3] For a set of predicates $\{P_i\}$, $\alpha(S)(x) = (\exists y : (\bigwedge i : P_i(x) \equiv P_i(y)) \wedge S(y))$.

3.4 Simple BDS's

The simplest case of a BDS is one where all global variables are of type $\mathbf{T}_0$, and the local variables are expressed by an array of type $\mathbf{T}_1 \mapsto \mathbf{T}_0$. This simple case is important, as it describes, for instance, the mutual exclusion protocol from Figure 1, and many multi-threaded programs, where the data structures that the threads operate on have a structure that is independent of the number of threads. For a simple BDS, the small model bounds are very small: 2 processes for a singly-quantified assertion ($\forall i : \theta(i)$), and 3 processes for a doubly-quantified assertion ($\forall i, j : i < j : \theta(i, j)$).

4 Experiments

This method has been implemented with TLV [40]. This implementation succeeds on several of the examples from [4]. Preliminary results are shown in Figure 2; details are at `http://www.cs.bell-labs.com/who/kedar/split-invariance`. The implementation generates quantified inductive assertions of the shape ($\forall i : \theta(i)$) and ($\forall i, j : i \neq j : \theta(i, j)$), denoted by 1-$\forall$ and 2-$\forall$, respectively.

The completeness of the split invariant calculation, especially the fact that it computes the *strongest* assertion, can be used to advantage. If the strongest split invariant θ^*_K calculated for an instance of size K fails the adequacy test, there can be no other θ' for which ($\forall i : \theta'(i)$) is inductive for all N. Thus, a failure indicates that there is no inductive invariant of the particular shape.

It is possible to dualize the theory presented here to compute "best" existential invariants, i.e., assertions of the form ($\exists i : \theta(i)$) where θ is the weakest formula for which the assertion is inductive and implies the correctness property. (This corresponds to the least/greatest fixpoint duality in the computation of the reachable states and $\mathsf{AG}(\phi)$.)

Some of the more difficult properties proved by the IIV tool require inductive assertions that are combinations of universal and existential properties. (The fact that something more than pure universal or existential invariants are required can be inferred by the failure to compute such a split invariant, as discussed above.) IIV uses a heuristic method (which may fail) to construct a Boolean combination of universal and existential assertions. It is not clear whether there is a complete and efficient method to construct such mixed invariants—our current implementation fails to prove these properties automatically. It is possible to replace (Skolemize) the existential quantifications with auxiliary program

Program	Invariant Type	Time(sec)	Bound	BDD size $\theta(1)$ or $\theta(1,2)$	Max. BDD
mutex	2-$\forall$	0.01	4	9	1327
mutex+	1-$\forall$	0.01	3	8	1007
szymanski	2-$\forall$	0.03	4	18	9956
token-ring	2-$\forall$	0.01	4	177	18341

Fig. 2. Results on the IIV examples (+ indicates introduction of auxiliary variables)

variables, leaving purely universal quantification that can be handled by the split-invariant method, but this requires a guess at the definition of the auxiliary variable. Devising a systematic procedure for introducing such variables is an important open question; ideas from the "environment abstraction" work [10] might be of help.

5 Related Work

There is a large body of work on parameterized verification and compositional analysis, the two subjects most closely connected with the topic of this paper.

The parameterized verification question is decidable for some classes: [21,17,15] are representative. These results are based on small model theorems for *temporal* properties. General methods for parameterized verification include those discussed in the introduction and others based on process summaries (cf. [36,31,41]) and acceleration methods based on automata-theory (cf. [30,1]). This earlier work does not consider completeness: the results on cutoffs, and the connection to inductive invariance are new contributions.

Compositional reasoning about concurrency goes back to the seminal work on non-interference by Owicki and Gries [38], extended to assume-guarantee reasoning by Chandy and Misra, and Jones [8,27,28]. (The book [12] has the history and technical relationships.) Flanagan and Qadeer apply the assume-guarantee approach to the verification of fixed instances of multi-threaded programs [20,18]. Assume-guarantee reasoning is combined with program abstraction in the BLAST tool [24]. These verification procedures are formulated for fixed-size instances, and do not, in general, lead to a correctness proof of a full parameterized system. The relationship between parameterized invariance, small model theorems, and compositional reasoning that forms the basis of the split-invariant method is also a new contribution.

Predicate abstraction [22] has been quite successful in deriving inductive invariants for non-parameterized programs [5,23]. Predicate abstraction is extended to derive quantified indexed predicates (typically for parameterized data structures) in [19,33]. This method is analyzed, and shown to be complete in [34], provided an appropriate indexed predicate set is given. The papers [19,33] give heuristics to determine this set, but the heuristics are not known to be complete.

It is worthwhile to compare the approaches based on predicate abstraction with indexed predicates and invisible/split invariants in a little more detail. Both have a common goal: to construct universally quantified invariants for parameterized programs. The indexed predicate method approaches this problem from "above", exploring a succession of abstract transition systems, all of which over-approximate the entire parameterized system. On the other hand, the invisible/split invariant method approaches this from "below", exploring successively larger instances of the parameterized system. Thus, split invariants always define states that are necessarily part of any inductive invariant, while indexed predicate exploration requires removal of non-inductive states. It is unclear which

method performs better in practice; but there may be fruitful ways of combining these different approximation approaches — this is the subject of ongoing work. Another interesting question for future work is whether the small model theorems in the BDS framework can be extended to apply to two examples identified in [33] that are outside the current class of BDS's; new results in [6] on array properties, and completeness results for (non-parameterized) predicate abstraction from [37,25,26] may apply here.

Acknowledgements. Thanks to the authors of the IIV tool [4], especially Ittai Balaban, for making their examples accessible. Thanks also to Ariel Cohen for discussions on split invariants, and for help with TLV. This research is supported, in part, by NSF grant CCR-0341658.

References

1. Parosh Aziz Abdulla, Ahmed Bouajjani, Bengt Jonsson, and Marcus Nilsson. Handling global conditions in parameterized system verification. In *CAV*, volume 1633 of *LNCS*, pages 134–145, 1999.
2. Krzysztof R. Apt and Dexter Kozen. Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.*, 22(6):307–309, 1986.
3. Tamarah Arons, Amir Pnueli, Sitvanit Ruah, Jiazhao Xu, and Lenore D. Zuck. Parameterized verification with automatically computed inductive assertions. In *CAV*, volume 2102 of *LNCS*, pages 221–234, 2001.
4. Ittai Balaban, Yi Fang, Amir Pnueli, and Lenore D. Zuck. IIV: An invisible invariant verifier. In *CAV*, volume 3576 of *LNCS*, pages 408–412, 2005.
5. T. Ball and S. K. Rajamani. The SLAM toolkit. In *CAV*, volume 2102 of *LNCS*. Springer Verlag, 2001.
6. Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. What's decidable about arrays? In *VMCAI*, volume 3855 of *LNCS*, pages 427–442, 2006.
7. Michael C. Browne, Edmund M. Clarke, and Orna Grumberg. Reasoning about networks with many identical finite state processes. *Inf. Comput.*, 81(1):13–31, 1989.
8. K.M. Chandy and J. Misra. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7(4), 1981.
9. Edmund M. Clarke and Orna Grumberg. Avoiding the state explosion problem in temporal logic model checking. In *PODC*, pages 294–303, 1987.
10. Edmund M. Clarke, Muralidhar Talupur, and Helmut Veith. Environment abstraction for parameterized verification. In *VMCAI*, volume 3855 of *LNCS*, pages 126–141, 2006.
11. D. Dams and K.S. Namjoshi. The existence of finite abstractions for branching time model checking. In *LICS*, 2004.
12. W-P. de Roever, F. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel, and J. Zwiers. *Concurrency Verification: Introduction to Compositional and Noncompositional Proof Methods*. Cambridge University Press, 2001.
13. E.W. Dijkstra. Guarded commands, nondeterminacy, and formal derivation of programs. *CACM*, 18(8), 1975.
14. E.W. Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics*. Springer Verlag, 1990.

15. E. Allen Emerson and Vineet Kahlon. Reducing model checking of the many to the few. In *CADE*, volume 1831 of *LNCS*, pages 236–254, 2000.
16. E. Allen Emerson and Kedar S. Namjoshi. Automatic verification of parameterized synchronous systems (extended abstract). In *CAV*, volume 1102 of *LNCS*, pages 87–98, 1996.
17. E.A. Emerson and K.S. Namjoshi. Reasoning about rings. In *ACM Symposium on Principles of Programming Languages*, 1995.
18. Cormac Flanagan, Stephen N. Freund, Shaz Qadeer, and Sanjit A. Seshia. Modular verification of multithreaded programs. *Theor. Comput. Sci.*, 338(1-3):153–183, 2005.
19. Cormac Flanagan and Shaz Qadeer. Predicate abstraction for software verification. In *POPL*, pages 191–202, 2002.
20. Cormac Flanagan and Shaz Qadeer. Thread-modular model checking. In *SPIN*, volume 2648 of *LNCS*, pages 213–224, 2003.
21. S. German and A.P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 1992.
22. S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *CAV*, volume 1254 of *LNCS*, 1997.
23. T. A. Henzinger, R. Jhala, R. Majumdar, G. C. Necula, G. Sutre, and W. Weimer. Temporal-safety proofs for systems code. In *CAV*, volume 2404 of *LNCS*, 2002.
24. Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Shaz Qadeer. Thread-modular abstraction refinement. In *CAV*, volume 2725 of *LNCS*, pages 262–274, 2003.
25. Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. Lazy abstraction. In *POPL*, pages 58–70, 2002.
26. Ranjit Jhala and Kenneth L. McMillan. A practical and complete approach to predicate refinement. In *TACAS*, volume 3920 of *LNCS*, pages 459–473, 2006.
27. C.B. Jones. *Development methods for computer programs including a notion of interference*. PhD thesis, Oxford University, 1981.
28. C.B. Jones. Tentative steps toward a development method for interfering programs. *ACM Trans. on Programming Languages and Systems (TOPLAS)*, 1983.
29. Y. Kesten and A. Pnueli. Verification by augmented finitary abstraction. *Information and Computation*, 163(1), 2000.
30. Yonit Kesten, Oded Maler, Monica Marcus, Amir Pnueli, and Elad Shahar. Symbolic model checking with rich ssertional languages. In *CAV*, volume 1254 of *LNCS*, pages 424–435, 1997.
31. Yonit Kesten and Amir Pnueli. Control and data abstraction: The cornerstones of practical formal verification. *STTT*, 2(4):328–342, 2000.
32. Robert P. Kurshan and Kenneth L. McMillan. A structural induction theorem for processes. In *PODC*, pages 239–247, 1989.
33. Shuvendu K. Lahiri and Randal E. Bryant. Constructing quantified invariants via predicate abstraction. In *VMCAI*, volume 2937 of *LNCS*, pages 267–281, 2004.
34. Shuvendu K. Lahiri and Randal E. Bryant. Indexed predicate discovery for unbounded system verification. In *CAV*, volume 3114 of *LNCS*, pages 135–147, 2004.
35. Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Eng.*, 3(2), 1977.
36. Ranko Lazic and David Nowak. A unifying approach to data-independence. In *CONCUR*, volume 1877 of *LNCS*, pages 581–595, 2000.
37. K. S. Namjoshi and R. P. Kurshan. Syntactic program transformations for automatic abstraction. In *CAV*, volume 1855 of *LNCS*. Springer Verlag, 2000.

38. Susan S. Owicki and David Gries. Verifying properties of parallel programs: An axiomatic approach. *Commun. ACM*, 19(5):279–285, 1976.
39. Amir Pnueli, Sitvanit Ruah, and Lenore D. Zuck. Automatic deductive verification with invisible invariants. In *TACAS*, volume 2031 of *LNCS*, pages 82–97, 2001.
40. Amir Pnueli and Elad Shahar. A platform for combining deductive with algorithmic verification. In *CAV*, volume 1102 of *LNCS*, pages 184–195, 1996. web: `www.cs.nyu.edu/acsys/tlv`.
41. Amir Pnueli, Elad Shahar, and Lenore D. Zuck. Network invariants in action. In *CONCUR*, volume 2421 of *LNCS*, pages 101–115, 2002.
42. Ichiro Suzuki. Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, 28(4):213–214, 1988.
43. Pierre Wolper and Vinciane Lovinfosse. Verifying properties of large sets of processes with network invariants. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 68–80, 1989.