Parameterized Verification with Automatically Computed Inductive Assertions*

T. Arons[1], A. Pnueli[1], S. Ruah[1], J. Xu[2], and L. Zuck[2]

[1] Weizmann Institute of Science, Rehovot, Israel
{tamarah,amir,sitvanit}@wisdom.weizmann.ac.il,
[2] New York University, New York, zuck@cs.nyu.edu

Abstract. The paper presents a method, called the method of *verification by invisible invariants*, for the automatic verification of a large class of parameterized systems. The method is based on the automatic calculation of candidate inductive assertions and checking for their inductiveness, using symbolic model-checking techniques for both tasks. First, we show how to use model-checking techniques over finite (and small) instances of the parameterized system in order to derive candidates for invariant assertions. Next, we show that the premises of the standard deductive INV rule for proving invariance properties can be automatically resolved by finite-state (BDD-based) methods with no need for interactive theorem proving. Combining the automatic computation of invariants with the automatic resolution of the VCs (verification conditions) yields a (necessarily) incomplete but fully automatic sound method for verifying large classes of parameterized systems. The generated invariants can be transferred to the VC-validation phase without ever been examined by the user, which explains why we refer to them as "invisible". The efficacy of the method is demonstrated by automatic verification of diverse parameterized systems in a fully automatic and efficient manner.

1 Introduction

The problem of *uniform verification of parameterized systems* is one of the most challenging problems in verification today. Given a parameterized system $S(N) : P[1] \| \cdots \| P[N]$ and a property p, uniform verification attempts to verify $S(N) \models p$ for every $N > 1$. Model checking is an excellent tool for *debugging* parameterized systems because, if the system fails to satisfy p, this failure can be observed for a specific (and usually small) value of N. However, once all the observable bugs have been removed, there remains the question whether the system is correct for all $N > 1$.

One method which can always be applied to verify parameterized systems is *deductive verification*. To verify that a parameterized system satisfies the invariance property $\Box\, p$, we may use rule INV presented in Fig. 1 [MP95a]. The

* This research was supported in part by the Minerva Center for Verification of Reactive Systems, a gift from Intel, a grant from the German - Israel Foundation for Scientific Research and Development, and ONR grant N00014-99-1-0131.

system to be verified is assumed to a transition system, with an assertion Θ describing the initial states, and a state transition relation ρ relating the values of (unprimed) variables in a state to the (primed) values of the variables in its successor. Premise I1 claims that the initial state of the system satisfies φ. Premise

$$\begin{array}{l} \text{I1. } \Theta \;\rightarrow\; \varphi \\ \text{I2. } \varphi \wedge \rho \;\rightarrow\; \varphi' \\ \text{I3. } \varphi \;\rightarrow\; p \\ \hline \quad\quad \Box\, p \end{array}$$

Fig. 1. The invariance Rule INV.

I2 claims that φ remains invariant under ρ. An assertion φ satisfying premises I1 and I2 is called *inductive*. Excluding the rare cases in which the property p is already inductive, the deductive verifier has to perform the following tasks:

1. Divine (invent) the auxiliary assertion φ.

2. Establish the validity of premises I1–I3.

Performing interactive first-order verification of implications such as the premises of rule INV for any non-trivial system is never an easy task. Neither is it a one-time task, since the process of developing the auxiliary invariants requires iterative verification trials, where failed efforts lead to correction of the previous candidate assertion into a new candidate.

In this paper we show that, for a wide class of parameterized systems, both of these tasks can be automated and performed directly by an appropriately enhanced model checker. The proposed method, called *verification by invisible invariants*, is based on the following idea: We start by computing the set of all reachable states of $S(N)$ for a sufficiently large N. We then project this set of states on one of the processes, say $P[1]$. Under the assumption that the system is sufficiently symmetric, we conclude that whatever is true of $P[1]$ will be true of all other processes. Thus, we abstract the characterization of all reachable states of process $P[1]$, denoted $\psi(1)$, into a generic $\psi(j)$ and propose the assertion $\varphi = \forall j : \psi(j)$ as a candidate for the inductive assertion which can be used within rule INV. To check that the candidate assertion φ is inductive and also implies the property p, we establish a small-model property which enables checking the premises of rule INV over $S(N_0)$ for a specific N_0 determined by the size of the local state space of a process in the system. The two tasks of calculating the candidate assertion φ and checking that it satisfies the premises of rule INV are performed automatically with no user interaction. This leads to the fact that the user never sees, or has to understand, the automatically produced inductive assertion. This explains the name of *verification by invisible invariants*.

The method of invisible invariants was first presented in [PRZ01], where it was used to verify a non-trivial cache protocol proposed by Steve German [Ger00]. The presentation in [PRZ01] allowed the method to be used only for a very restricted class of systems. The main limitations were that the only predicates allowed in this class were equality comparisons between parameterized types, and the only arrays were of type $[1..N] \mapsto \textbf{bool}$. In this paper, we extend the applicability of the method in several dimensions as follows:

- Allowing inequality comparisons of the form $u < v$ between parameterized types and operations such as $u + 1$ and $u \oplus 1$ (incrementation modulo N).
- Allowing several parameterized types and arrays that map one parameterized type to another.

These extensions significantly broaden the scope of applicability of the method, allowing us to deal with diverse examples such as various cache protocols, a 3-stage pipeline, Szymanski's algorithm for mutual exclusion, a token-ring algorithm, a restricted form of the Bakery algorithm, and an N-process version of Peterson's algorithm for mutual exclusion, all in a fully automatic and efficient manner.

Related Work. The problem of uniform verification of parameterized systems is, in general, undecidable [AK86]. There are two possible remedies to this situation: either we should look for restricted families of parameterized systems for which the problem becomes decidable, or devise methods which are sound but, necessarily incomplete, and hope that the system of interest will yield to one of these methods.

Among the representatives of the first approach we can count the work of German and Sistla [GS92] which assumes a parameterized system where processes communicate synchronously, and shows how to verify single-index properties. Similarly, Emerson and Namjoshi provided a decision procedure for proving a restricted set of properties on ring algortihms [EN95], and proved a PSPACE complete algorithm for verification of synchronously communicating processes [EN96]. Many of these methods fail when we move to asynchronous systems where processes communicate by shared variables. Perhaps the most advanced of this approach is the paper [EK00] which considers a general parameterized system allowing several different classes of processes. However, this work provides separate algorithms for the cases that the guards are either all disjunctive or all conjunctive. A protocol such as the cache example we consider in [PRZ01] which contains some disjunctive and some conjunctive guards, cannot be handled by the methods of [EK00].

The sound but incomplete methods include methods based on explicit induction ([EN95]) network invariants, which can be viewed as implicit induction ([KM95], [WL89], [HLR92], [LHR97]), methods that can be viewed as abstraction and approximation of network invariants ([BCG86], [SG89], [CGJ95], [KP00]), and other methods that can be viewed as based on abstraction ([ID96]). The papers in [CR99a,CR99b,CR00] use structural induction based on the notion of a network invariant but significantly enhance its range of applicability by using a generalization of the data-independence approach which provides a powerful abstraction capability, allowing it to handle network with parameterized topologies. Most of these methods require the user to provide auxiliary constructs, such as a network invariant or an abstraction mapping. Other attempts to verify parameterized protocols such as Burn's protocol [JL98] and Szymanski's algorithm [GZ98,MAB+94] relied on abstraction functions or lemmas provided by the user. The work in [LS97] deals with the verification of safety properties of parameterized networks by abstracting the behavior of the system. PVS ([SOR93]) is used to discharge the generated VCs.

Among the automatic incomplete approaches, we should mention the methods relying on "regular model-checking" [KMM+97,ABJN99,JN00,PS00], where a class of systems which include our bounded-data systems as a special case is analyzed representing linear configurations of processes as a word in a regular language. Unfortunately, many of the systems analyzed by this method cause the analysis procedure to diverge and special *acceleration* procedures have to be applied which, again, requires user ingenuity and intervention.

The works in [ES96,ES97,CEFJ96,GS97] study symmetry reduction in order to deal with state explosion. The work in [ID96] detects symmetries by inspection of the system description. Closer in spirit to our work is the work of McMillan on compositional model-checking (e.g. [McM98b]), which combines automatic abstraction with finite-instantiation due to symmetry.

2 The Systems We Consider

Let $\mathbf{type}_0$ denote the set of boolean and fixed (unparameterized) finite-range basic types which, for simplicity, we often denote as **bool**. Let $\mathbf{type}_1, \ldots, \mathbf{type}_m$ be a set of basic parameterized types, where each $\mathbf{type}_i$ includes integers in the range $[1..N_i]$ for some $N_i \in \mathbb{N}^+$. The systems we study include variables that are either $\mathbf{type}_i$ variables, for some $i \in \{0, \ldots, m\}$, or arrays of the type $\mathbf{type}_i \mapsto \mathbf{type}_j$ for $i > 0, j \geq 0$. For a system that includes types $\mathbf{type}_1, \ldots, \mathbf{type}_k$, we refer to $N_1, \ldots, N_k$ as the *system's parameters*. Systems are distinguished by their *signatures*, which determine the types of variables allowed, as well as the assertions allowed in the transition relation and the initial condition. Whenever the signature of a system includes the type $\mathbf{type}_i \mapsto \mathbf{type}_j$, we assume by default that it also includes the types $\mathbf{type}_i$ and $\mathbf{type}_j$.

Atomic formulae may compare two expressions of the same type, where the only allowed expressions are a variable y or an array reference $z[y]$. Thus, if y and $\widetilde{y}$ are $\mathbf{type}_i$ variables, then $y \leq \widetilde{y}$ is an atomic formula, and so are $z[y] \leq z[\widetilde{y}]$, $x \leq z[y]$, and $z[y] \leq x$ for an array $z\colon \mathbf{type}_i \mapsto \mathbf{type}_j$ and $x\colon \mathbf{type}_j$.

Formulae, used in the transition relation and the initial condition, are obtained from the atomic formulae by closing them under negation, disjunction, and existential quantifiers, for appropriately typed quantifiers.

A *bounded-data discrete system* (BDS) $S = \langle V, \Theta, \rho \rangle$ consists of

- V–A set of *system variables*, as described above. A *state* of the system S provides a type-consistent interpretation of the system variables V. For a state s and a system variable $v \in V$, we denote by $s[v]$ the value assigned to v by the state s. Let Σ denote the set of states over V.
- $\Theta(V)$–The *initial condition*: A formula characterizing the initial states.
- $\rho(V, V')$–The *transition relation*: A formula, relating the values V of the variables in state $s \in \Sigma$ to the values V' in an S-successor state $s' \in \Sigma$.

For all the systems we consider here, we assume that the transition relation can be written as

$$\exists \underbrace{h_1^1,\ldots,h_{H_1}^1}_{\mathbf{type}_1},\ldots,\underbrace{h_1^k,\ldots,h_{H_k}^k}_{\mathbf{type}_k} \forall \underbrace{t_1^1,\ldots,t_{T_1}^1}_{\mathbf{type}_1},\ldots,\underbrace{t_1^k,\ldots,t_{T_k}^k}_{\mathbf{type}_k} : R(\vec{h},\vec{t})) \tag{1}$$

where $R(\vec{h},\vec{t})$ is a well-typed quantifier-free formula. It follows that every BDS is associated with $H_1,\ldots,H_K$ and $T_1,\ldots,T_K$.

Note that Θ and ρ are restricted to "formulae" defined in the previous section. Since in this paper we only consider the verification of invariance properties, we omitted from the definition of a BDS the components that relate to fairness. When we will work on the extension of these methods to liveness, we will add the relevant fairness components.

A computation of the BDS $S = \langle V, \Theta, \rho \rangle$ is an infinite sequence of states $\sigma : s_0, s_1, s_2, ...$, satisfying the requirements:

- *Initiality* — s_0 is initial, i.e., $s_0 \models \Theta$.
- *Consecution* — For each $\ell = 0, 1, ...$, the state $s_{\ell+1}$ is a S-successor of s_ℓ. That is, $\langle s_\ell, s_{\ell+1} \rangle \models \rho(V, V')$ where, for each $v \in V$, we interpret v as $s_\ell[v]$ and v' as $s_{\ell+1}[v]$.

Mainly, we consider systems with signature $\langle \mathbf{type}_1 \mapsto \mathbf{type}_0, \mathbf{type}_1 \mapsto \mathbf{type}_2 \rangle$. This signature admits arrays which are subscripted by $\mathbf{type}_1$-elements and range over either $\mathbf{type}_0$ or $\mathbf{type}_2$. We name the variables in such a system as follows: $x_1,\ldots,x_a$ are $\mathbf{type}_0$ variables, $y_1,\ldots,y_b$ are $\mathbf{type}_1$ variables, $z_1,\ldots,z_c$ are arrays of type $\mathbf{type}_1 \mapsto \mathbf{type}_0$, $u_1,\ldots,u_d$ are $\mathbf{type}_2$ variables, and $w_1,\ldots,w_e$ are arrays of type $\mathbf{type}_1 \mapsto \mathbf{type}_2$.

We keep these naming convention for systems with simpler signatures. Thus, a system with no $\mathbf{type}_2$ variables will have only x-, y-, or z-variables. We assume that the description of each system contains a z-variable π that includes the location of each process.

3 The Method of Invisible Invariants

Our goal is to provide an automated procedure to generate proofs according to INV. While in general the problem is undecidable [AK86], we offer a heuristic that had proven successful in many cases for the systems we study, where the strengthening assertions are of the form $\forall i_1,\ldots,i_I : \psi(\vec{i})$ where $i_1,\ldots,i_I$ are all mutually distinct typed variables, and $\psi(\vec{i})$ is a quantifier-free formula. We elaborate the method for the case of systems with signature $\langle \mathbf{type}_1 \mapsto \mathbf{type}_0, \mathbf{type}_1 \mapsto \mathbf{type}_2 \rangle$ as defined in Section 2. Thus, we are seeking an assertion of the type $\forall i_1^1,\ldots,i_{I_1}^1, i_1^2,\ldots,i_{I_2}^2 : \psi(\vec{i^1},\vec{i^2})$ where for $i_1^\ell,\ldots,i_{I_\ell}^\ell$ are all mutually distinct $\mathbf{type}_\ell$ variables for $\ell = 1, 2$, and $\psi(\vec{i^1},\vec{i^2})$ is a quantifier-free formula. In the next sections we obtain (small) bounds for the parameters of this family of systems, such that it suffices to prove the premises of INV on systems whose parameters are bounded by those bounds. This offers a decision procedure for the premises of rule INV, which greatly simplifies the process of deductive verification. Yet, it still leaves open the task of inventing the strengthening assertion φ, which may become quite complex for all but the simplest systems. In this section we propose a heuristic for an algorithmic construction of an inductive assertion for a given BDS. In particular, we provide an algorithm to construct an inductive assertion of the form we are seeking for a two-parameter systems $S(N_1^0, N_2^0)$, where N_1^0 and N_2^0 are the bounds computed for the system.

1. Let *reach* be the assertion characterizing all the reachable states of system $S(N_1^0, N_2^0)$. Since $S(N_1^0, N_2^0)$ is finite-state, *reach* can be computed by standard model-checking techniques.
2. Let ψ_{I_1,I_2} be the assertion obtained from *reach* by projecting away all the references to $\mathbf{type}_1$ values other than $1,\ldots,I_1$, and $\mathbf{type}_2$ values other than $1,\ldots,I_2$.
3. Let $\psi(\vec{i^1},\vec{i^2})$ be the assertion obtained from ψ_{I_1,I_2} by *abstraction*, which involves the following transformations: For every $j=1,\ldots,I_1$ and and $k=1,\ldots,I_2$, replace any reference to $z_r[j]$ by a reference to $z_r[i^1_j]$, any reference to $w_r[j]=k$ (resp. $w_r[j]\neq k$) by a reference to $w_r[i^1_j]=i^2_k$ (resp. $w_r[i^1_j]\neq i^2_k$), any sub-formula of the form $y_r=j$, $j\leq I_1$ by the formula $y_r=i^1_j$, any sub-formula of the form $y_r=v$ for $v>I_1$ by the formula $\bigwedge_{j=1}^{I_1} y_r\neq i^1_j$, any sub-formula of the form $u_r=k$, $k\leq I_2$, by the formula $u_r=i^2_k$, and any sub-formula of the form $u_r=v$ for $v>I_2$ by the formula $\bigwedge_{k=1}^{I_2} u_r\neq i^2_k$.
4. Let $\varphi := \bigwedge_{1\leq i^1_1<\ldots<i^1_{I_1}\leq N^0_1, 1\leq i^2_1<\ldots<i^2_{I_2}\leq N^0_2} \psi(\vec{i^1},\vec{i^2})$.
5. Check that φ is inductive over $S(N_1^0, N_2^0)$.
6. Check that $\varphi \rightarrow p$ is valid.

If tests (5) and (6) both yield a positive result, then the property p has been verified. The procedure described here is a generalization of a similar procedure in [PRZ01].

4 Obtaining the Bounds

In this section we obtain (small) bounds for the parameters of various systems according to their signatures, and show that it suffices to prove the premises of INV on systems whose parameters are within these bounds. We first present our main claim, which establishes the bounds for the most general system.

Consider a BDS $S(N_1,N_2)$ with signature $\langle \mathbf{type}_1 \mapsto \mathbf{bool}, \mathbf{type}_1 \mapsto \mathbf{type}_2\rangle$ to which we wish to apply proof rule INV with the assertions φ and p having each the form $\forall i^1_1,\ldots,i^1_{I_1},i^2_1,\ldots,i^2_{I_2} : \psi(\vec{i^1},\vec{i^2})$, where every i^1_j (resp. i^2_r) is a $\mathbf{type}_1$ (resp. $\mathbf{type}_2$) variable, and $\psi(\vec{i^1},\vec{i^2})$ is a quantifier free formula. The transition relation of the system is described by equation (1) with $k=2$.

Consider a state s of the system $S(N_1,N_2)$. The *size* of s is (N_1,N_2). We say that (N_1,N_2) is smaller than (N'_1,N'_2), and denote it by $(N_1,N_2)\preceq(N'_1,N'_2)$ if $N_1\leq N'_1$ and $N_2\leq N'_2$.

Lemma 1. *The premises of rule* INV *are valid over* $S(N_1,N_2)$ *for all* $(N_1,N_2)\succeq(2,2)$ *iff they are valid over* $S(N_1,N_2)$ *for all* $(N_1,N_2)\preceq(N_1^0,N_2^0)$ *where* $N_1^0=b+I_1+H_1$ *and* $N_2^0=d+I_2+H_2+e(b+I_1+H_1)$.

Proof. The most complex verification condition in rule INV is premise (I2) which can be written as: $(\forall \vec{j} : \psi(\vec{j})) \ \wedge \ (\exists \vec{h} \forall \vec{t} : R(\vec{h}, \vec{t})) \rightarrow \forall \vec{i} : \psi'(\vec{i})$ To prove the claim, it suffices to show that if the formula

$$(\forall \vec{j} : \psi(\vec{j})) \ \wedge \ (\forall \vec{t} : R(\vec{h}, \vec{t})) \ \wedge \ \neg\psi'(\vec{i}) \qquad (2)$$

is satisfiable over a state of size $(N_1, N_2) \succeq (2, 2)$, it is also satisfiable over a state of size $(\alpha_1, \alpha_2) \preceq (N_1^0, N_2^0)$.

Let s be a state of size $(N_1, N_2) \succeq (2, 2)$ which satisfies assertion (2). Let $Val_1 \subseteq [1..N_1]$ be the set of (pairwise) distinct values the state s assigns to the variables $V_{aug}^1 = \{h_1^1, \ldots, h_{_{H_1}}^1, i_1^1, \ldots, i_{_{I_1}}^1, y_1, \ldots, y_b\}$. Let $\alpha_1 = |\,Val_1|$; obviously, $\alpha_1 \leq H_1 + I_1 + b = N_1^0$ and $\alpha_1 \leq N_1$. Similarly, let $Val_2 \subseteq [1..N_2]$ be the set of (pairwise) distinct values the state s assigns to the variables

$$V_{aug}^2 \ = \ \{h_1^2, \ldots, h_{_{H_2}}^2, i_1^2, \ldots, i_{_{I_2}}^2, u_1, \ldots, u_d\} \cup \{w_\ell[k] : \ell = 1, \ldots, e, \ k \in Val_1\}$$

Let $\alpha_2 = |\,Val_2|$; obviously, $\alpha_2 \leq H_2 + I_2 + d + e(H_1 + I_1 + b) = N_2^0$ and $\alpha_2 \leq N_2$.

For every $\ell = 1, 2$, assume the distinct $\mathbf{type}_\ell$ values are sorted $v_1^\ell < v_2^\ell < \cdots < v_{\alpha_\ell}^\ell$. Let Π_ℓ be a permutation on $[1..N_\ell]$ such that for every $j = 1, \ldots, \alpha_\ell$, $\Pi_\ell^{-1}(v_j^\ell) = j$. The two permutations, Π_1, and Π_2, help construct a new state where the range of values assigned to the variables in V_{aug}^1 (resp. V_{aug}^2) is reduced from $[1..N_1]$ (resp. $[1..N_2]$) to $[1..\alpha_1]$ (resp. $[1..\alpha_2]$).

Consider now a state $\widetilde{s}$ of size (α_1, α_2), where: $\widetilde{x}_r = x_r$ for every $r \in [1..a]$, $\widetilde{y}_r = \Pi_1^{-1}(y_r)$ for every $r \in [1..b]$, $\widetilde{z}_r[h] = z_r[\Pi_1(k)]$ for every $r \in [1..c]$ and $k \in [1..\alpha_1]$, $\widetilde{u}_r = \Pi_2^{-1}(u_r)$ for every $r \in [1..d]$, and $\widetilde{w}_r[k] = \Pi_2^{-1}(w_r[\Pi_1(k)])$ for every $r \in [1..e]$ and $k \in [1..\alpha_1]$.

It is easy to establish, by induction on the structure of the quantifier-free formulae ψ and R, that the evaluation of formula (2) over $\widetilde{s}$ yields the same truth values as the evaluation of formula (2) over s. Consequently, $\widetilde{s}$ is a state of size (α_1, α_2) that satisfies formula (2). $\square$

The Class $\langle \mathbf{type}_1 \mapsto \mathbf{bool} \rangle$ This is the class of systems which have boolean and other finite-domain parameterized arrays. The algorithms belonging to this class are MUX-SEM (mutual exclusion by semaphores), a 3-stages pipeline [BD94], [McM98a], Steve German's cache [Ger00,PRZ01], and the Illinois' Cache Algorithm [PP84,Del00], all studied in [PRZ01]. In addition, it includes Szymanski's mutual exclusion algorithm [Szy88] and token ring algorithms.

This class extends the class of systems considered in [PRZ01], which only allowed comparison for equality between two y_r variables, by allowing tests for inequalities. Since there are no $\mathbf{type}_2$ variables in the system, an immediate consequence of Lemma 1 is:

Corollary 1. *Let $S(N)$ be a parameterized system of signature $\langle \mathbf{type}_1 \mapsto \mathbf{bool} \rangle$ to which we wish to apply proof rule* INV *with the assertions φ and p having each the form $\forall i_1, \ldots, i_{_I} : \psi(i_1, \ldots, i_{_I})$. Then, the premises of rule* INV *are valid over $S(N)$ for all $N > 1$ iff they are valid over $S(N)$ for all N, $1 < N \leq N_0$, where $N_0 = b + I + H$.*

In Fig. 2, we present a mutual exclusion algorithm due to B. Szymanski [Szy88], which uses inequality comparisons between process indices. In the system, $b = 0$

in N : **integer where** $N > 1$
local zw, zs : **array** $[1..N]$ **of boolean where** $zw = zs = 0$

$\overset{N}{\underset{i=1}{\Big\|}} P[i] ::$
loop forever do
- ℓ_0 : **NonCritical**
- ℓ_1 : **await** $\forall j : \neg zs[j]$
- ℓ_2 : $(zw[i], zs[i]) := (1, 1)$
- ℓ_3 : **If** $\exists j : at\_\ell_{1,2}[j]$
  - **then** $zs[i] := 0$; **go-to** ℓ_4
  - **else** $zw[i] := 0$; **go-to** ℓ_5
- ℓ_4 : **await** $\exists j : zs[j] \wedge \neg zw[j]$ **then** $(zw[i], zs[i]) := (0, 1)$
- ℓ_5 : **await** $\forall j : \neg zw[j]$
- ℓ_6 : **await** $\forall j : j < i : \neg zs[j] \wedge \neg zw[j]$
- ℓ_7 : **Critical**
- ℓ_8 : $zs[i] := 0$

Fig. 2. Parameterized mutual exclusion Algorithm SZYMANSKI

and $H = 2$. To apply this claim for system SZYMANSKI, where the property to be verified is mutual exclusion, which can be specified by $p: \quad \forall i \neq j : \neg(at\_\ell_7[i] \;\wedge\; at\_\ell_7[j])$, we set $I = 2$, which led to a cutoff value of $N_0 = 4$.

Transition Relations with "+1" or"$\oplus$1" Constrains: Some of the parameterized systems which we wish to verify have atomic sub-formulae of the forms $h_2 = h_1 + 1$ or $h_2 = h_1 \oplus 1$ (which stands for $h_2 = (h_1 \; mod \; N) + 1$) within their transition relations. We resolve this difficulty by observing that

$$\exists h_1, h_2 : h_2 = h_1 + 1 \;\wedge\; \forall \vec{t} : R(h_1, h_2, \vec{t}) \;\leftrightarrow$$
$$\exists h_1, h_2 : h_1 < h_2 \;\wedge\; (\forall t : t \leq h_1 \;\vee\; h_2 \leq t) \;\wedge\; \forall \vec{t} : R(h_1, h_2, \vec{t})$$
$$\exists h_1, h_2 : h_2 = h_1 \oplus 1 \;\wedge\; \forall \vec{t} : R(h_1, h_2, \vec{t}) \;\leftrightarrow\; \exists h_1, h_2 : \Big((h_1 < h_2 \;\wedge\; \forall t : t \leq h_1 \;\vee$$
$$h_2 \leq t) \;\vee\; (h_2 < h_1 \;\wedge\; \forall t : h_2 \leq t \leq h_1) \Big) \;\wedge\; \forall \vec{t} : R(h_1, h_2, \vec{t})$$

In the first translation, we ensure that $h_2 = h_1 + 1$ by requiring that h_1 be smaller than h_2 and that, for every other index t, either t is smaller or equal to h_1 or it is greater or equal to h_2. In the second translation, expected to capture the constraint $h_2 = h_1 \oplus 1$, we repeat the characterization of $h_2 = h_1 + 1$ but also allow the option that $h_1 = N$ and $h_2 = 1$. This is ensured by $h_2 < h_1 \wedge \forall t : h_2 \leq t \leq h_1$. Since $(\forall t : P(t) \;\vee\; \forall t : Q(t)) \leftrightarrow \forall t_1, t_2 : (P(t_1) \;\vee\; Q(t_2))$, the formulae above can be easily expressed in the form required for transition relation. Thus, the cutoff value established in Corollary 1 is still valid for both these cases.

in N : **integer where** $N > 1$
local $token$: $[1..N]$

$\overset{N}{\underset{i=1}{\Big\|}} C[i] ::$
loop forever do
- ℓ_0 : **NonCritical**
- ℓ_1 : **await** $i = token$
- ℓ_2 : **Critical**

$\|$

$\overset{N}{\underset{i=1}{\Big\|}} T[i] ::$
loop forever do
- m_0 : **when** $at\_\ell_0[i] \wedge token = i$
  - $token := i \oplus 1$

Fig. 3. Parameterized mutual exclusion Algorithm TOKEN-RING

In Fig. 3, we present a program which coordinates mutual exclusion by passing a token around a ring. The signature of the system is $\langle \mathbf{type}_1 \mapsto \mathbf{bool} \rangle$. The transition relation for this program includes the atomic formula $h_2 = h_1 \oplus 1$. The program consists of N client processes $C[1], \ldots, C[N]$ which can enter their critical section only when they have the token. Process $C[i]$ has the token when the token variable *token* equals i. In addition, there are N *transmission* processes such that process $T[i]$ is responsible for moving the token from client $C[i]$ to client $C[i \oplus 1]$ whenever process $C[i]$ is in its non-critical section. For this program we have the parameters $b = 1$ (a single $[1..N]$-variable: *token*), and $H = 2$. According to Corollary 1, to establish an inductive assertion of the form $\forall i_1, i_2 : \psi(i_1, i_2)$ for program TOKEN-RING, it suffices to take a cutoff value of $N_0 = 5$.

The Class $\langle \mathbf{type}_1 \mapsto \mathbf{bool}, \mathbf{type}_1 \mapsto \mathbf{type}_2 \rangle$ In Fig. 4, we present a program which implements a restricted version of the Bakery Algorithm by Lamport.

$$
\begin{array}{l}
\quad\quad\quad \textbf{in} \quad N_1, N_2 : \textbf{integer where } N_1 > 1, N_2 > 1 \\
\quad\quad\quad \textbf{local} \quad w \quad : \textbf{array } [1..N_1] \textbf{ of } [1..N_2] \textbf{ where } w = N_1 \\
\quad\quad\quad \quad\quad\quad\;\; z \quad : \textbf{array } [1..N_1] \textbf{ of boolean where } z = 0 \\
\displaystyle\prod_{i=1}^{N_1} C[i] :: \left[\begin{array}{l}
\textbf{loop forever do} \\
\left[\begin{array}{l}
\ell_0 : \textbf{NonCritical} \\
\ell_1 : \displaystyle\bigvee_{u=1}^{N_2} \textbf{when } \forall j : (\neg z[j] \;\vee\; u > w[j]) \textbf{ do } (z[i], w[i]) := (1, u) \\
\ell_2 : \textbf{await } \forall j : (\neg z[j] \;\vee\; w[i] < w[j]) \\
\ell_3 : \textbf{Critical} \\
\ell_4 : z[i] := 0
\end{array}\right]
\end{array}\right] \\
\quad\quad \| \\
\displaystyle\prod_{i=1}^{N_1} R[i] :: \left[\begin{array}{l}
\textbf{loop forever do} \\
m_0 : \displaystyle\bigvee_{u=1}^{N_2} \left[\begin{array}{l}
\textbf{when } z[i] \;\wedge\; \forall j : (\neg z[j] \;\vee\; w[j] < u \;\vee\; w[j] \geq w[i]) \textbf{ do} \\
\quad w[i] := u
\end{array}\right]
\end{array}\right]
\end{array}
$$

Fig. 4. Parameterized mutual exclusion Algorithm BAKERY

In the standard Bakery algorithm the variables $w[i]$ are unbounded natural numbers. Here we bound them by N_2. To make sure that they do not get stuck at N_2 and prevent any new values to be drawn at statement ℓ_1, we have the *reducing process* $R[i]$ which attempts to identify a *gap* just below the current value of $w[i]$. Such a gap is a positive natural number u smaller than $w[i]$ and which is not currently occupied by any $w[j]$ for an active $C[j]$, and such that all active $w[j]$ are either below u or above $w[i]$. Client $C[j]$ is considered active if $z[j] = 1$. On identifying such a gap u, process $R[i]$ *reduces* $w[i]$ to u. The disjunction $\bigvee_{u=1}^{N_2}$ occurring at statements ℓ_1 and m_0 denotes a non-deterministic choice over all possible values of u in the range $[1..N_2]$, provided the chosen value of u satisfies the condition appearing in the enclosed **when** statement.

The property of mutual exclusion, it can be written as $p : \forall i \neq j : \neg(at\_\ell_3[i] \wedge at\_\ell_3[j])$. Since both i and j are of type $\mathbf{type}_1$, this leads to a choice of $I_1 = 2$ and $I_2 = 0$. From the program we can conclude that $b = 0$, $d = 0$, and $e = 1$ (corresponding to the single $[1..N_1] \mapsto [1..N_2]$ array w). The transition relation can be written in the form $\exists i, u : \forall \vec{t} : R$, leading to $H_1 = 1$ (the auxiliary variable

i) and $H_2 = 1$ (the auxiliary variable $u : \mathbf{type}_2$). Using these numbers, we obtain a cutoff value pair $(N_1^0, N_2^0) = (3, 4)$.

Arbitrary Stratified Systems Lemma 1 can be generalized to systems with arbitrary array types, as long as the type structure is *stratified*, i.e., $i < j$ for each type $\mathbf{type}_i \mapsto \mathbf{type}_j$. Consider a stratified BDS with k parameterized types $\mathbf{type}_1, \ldots, \mathbf{type}_k$. Let b_i be the number of $\mathbf{type}_i$ vairables in the system, and let e_{ij} be the number of $\mathbf{type}_i \mapsto \mathbf{type}_j$ arrays in the system.

Corollary 2. *Let S be a k-parameter BDS with $k \geq 1$ stratified types to which we wish to apply proof rule* INV *with the assertions φ and p having each the form $\forall i_1^1, \ldots, i_{I_1}^1, \ldots, i_1^k, \ldots, i_{I_k}^k : \psi(\vec{i})$. Then, the premises of rule* INV *are valid over $S(N_1, \ldots, N_k)$ for all $N_1, \ldots, N_k > 1$ iff they are valid over $S(N_1^0, \ldots, N_k^0)$ where $N_1^0 = b_1 + H_1 + I_1$, and for every $i = 2, \ldots, k$, $N_i^0 = (b_i + H_i + I_i) + \sum_{j=1}^{i-1}(e_{ji} \cdot N_j^0)$.*

5 Systems with Unstratified Array Structure

```
              in     N      : integer where N > 1
              type   Pr_id  : [1..N]
                     Level  : [0..N]
              local  y      : array Pr_id of Level where y = 0
                     s      : array Level of Pr_id
   N                ┌ loop forever do
  ||   P[i] ::      │  ┌ ℓ0 : NonCritical
 i=1                │  │ ℓ1 : (y[i], s[1]) := (1, i)
                    │  │ ℓ2 : while y[i] < N do
                    │  │       [ ℓ3 : await s[y[i]] ≠ i ∨ ∀j ≠ i : y[j] < y[i] ]
                    │  │       [ ℓ4 : (y[i], s[y[i] + 1]) := (y[i] + 1, i)     ]
                    │  │ ℓ5 : Critical
                    └  └ ℓ6 : y[i] := 0
```

Fig. 5. Parameterized mutual exclusion Algorithm PETERSON

There are many interesting systems for which the restriction of stratification does not apply. For example, consider program PETERSON presented in Fig. 5, which implements a mutual exclusion algorithm due to Peterson. Obviously, this system has an unstratified array structure.

When the system has an unstratified array structure, we lose the capability of reducing any counter-model which violates $(\forall \vec{j} : \psi(\vec{j})) \ \wedge \ (\exists \vec{h} \forall \vec{t} : R(\vec{h}, \vec{t})) \rightarrow \forall \vec{i} : \psi'(\vec{i})$ to a model of size not exceeding N_0. But this does not imply that we cannot resolve this verification condition algorithmically. The first step in any deductive proof of a formula such as the above formula is that of *skolemization* which removes all existential quantifications on the left-hand side and all universal quantifications on the right-hand side of the implication, leading to

$$(\forall \vec{j} : \psi(\vec{j})) \ \wedge \ (\forall \vec{t} : R(\vec{h}, \vec{t})) \quad \rightarrow \quad \psi'(\vec{i}) \tag{3}$$

In subsequent steps, the deductive proof instantiates the remaining universal quantifications for $\vec{j}$ and $\vec{t}$ by concrete terms. Most often these concrete terms are taken from the (now) free variables of (3), namely, $\vec{h}$ and $\vec{i}$. Inspired by this standard process pursued in deductive verification, we suggest to replace Formula (3) by

$$(\bigwedge_{\vec{j}\in\{\vec{h},\vec{i}\}} \psi(\vec{j})) \wedge (\bigwedge_{\vec{t}\in\{\vec{h},\vec{i}\}} R(\vec{h},\vec{t})) \quad\rightarrow\quad \psi'(\vec{i}), \tag{4}$$

which is obtained by replacing the universal quantification over $\vec{j}$ and $\vec{t}$ by a conjunction in which each conjunct is obtained by instantiating the relevant variables ($\vec{j}$ or $\vec{t}$) by a subset (allowing replication) of the free variables $\vec{h}$ and $\vec{i}$. The conjunction should be taken over all such possible instantiations. The resulting quantifier-free formula is not equivalent to the original formula (3) but the validity of (4) implies the validity of (3). For a quantifier-free formula such as (4), we have again the property of model reduction, which we utilize for formulating the appropriate decision procedure for unstratified systems.

Consider an unstratified system $S(N)$ with b variables of type $[1..N]$ and e arrays of type $[1..N] \mapsto [1..N]$. As before, let H denote the number of existentially quantified variables in the definition of ρ and let I denote the number of universally quantified variables in the definition of φ. Furthermore, assume that the transition relation or candidate assertion do not contain nested arrays references. For example, we will replace the formula $s[y[i]] \neq i$ by $y[i] = h \wedge s[h] \neq i$, where h is a fresh auxiliary variable. Let $\text{INV}^{\wedge}$ denote a version of rule INV in which all premises have been skolemized first and then, the remaining universal quantifications replaced by conjunctions over all instantiations by the free variables in each formula.

Claim. Let $S(N)$ be a parameterized system as described above. Then if $S(N_0)$ satisfies the premises of rule INV applied to property p for $N_0 = (e+1)(b+I+H)$, we can conclude that p is an invariant of $S(N)$ for every $N > 1$.

For strongly typed systems, such as PETERSON, where comparisons and assignments are only allowed between elements of the same type, we can provide more precise bounds. Assume that the system has two types and that each of the bounds can be split into two components. Then the bound on N_0 can be refined into $N_0 = \max(b_1 + I_1 + H_1 + e_{21}(b_2 + I_2 + H_2), b_2 + I_2 + H_2 + e_{12}(b_1 + I_1 + H_1))$, where e_{21} and e_{12} denote the number of $\mathbf{type}_1 \mapsto \mathbf{type}_2$ and $\mathbf{type}_2 \mapsto \mathbf{type}_1$-arrays. For the case of PETERSON, we have $b_1 = b_2 = 0$, $I_1 = I_2 = 2$, $H_1 = 1$, $H_2 = 2$, and $e_{12} = e_{21} = 1$, which leads to $N_0 = 7$.

6 The Proof of the Pudding

According to a common saying "the proof of the pudding is in the eating". In this section, we present the experimental results obtained by applying the method of *invisible invariants* to various systems. Table 1 summarizes these results.

The second column of the table specifies the number of processes used in the verification process. In some cases, we took a value higher than the required minimum. The τ_1 column specifies the time (in seconds) it took to compute the reachable states. Column τ_2 specifies the time it took to compute the candidate inductive assertion. Finally, column τ_3 specifies the time it took to check the premises of rule INV.

The systems on the left are each a single-type system which only employs equality comparison in their transition relations and candidate assertions. SZYMANSKI employs inequalities, and TOKEN-RING needs the relation $h_2 = h_1 \oplus 1$ in its transition relation. BAKERY is a stratified two-type system employing inequality comparisons, and PETERSON is an unstratified two-type system. To obtain inductiveness in the Illinois' cache protocol we had to add an auxiliary variable called *last_dirty* which records the index of the last process which made its cache entry dirty.

System	N_0	τ_1	τ_2	τ_3
MUX-SEM	5	.01	.01	.01
S. German's Cache	4	10.21	10.72	133.04
Illinois's Cache	4	1.47	.04	.58
3-stages Pipeline	6	20.66	.27	29.59

System	N_0	τ_1	τ_2	τ_3
SZYMANSKI	4	$< .01$	.06	.06
TOKEN-RING	5	$< .01$	$< .01$	$< .01$
BAKERY	5	.41	.16	.25
PETERSON	(6,7)	79	1211	240

Table 1. Summary of experimental results.

7 Conclusion and Future Work

The paper studies the problem of uniform verification of parameterized systems. We have introduced the method of verification by invisible invariants–a heuristic that has proven successful for fully automatic verification of safety properties for many parameterized systems.

We are currently working on extending the method so that it also encompasses liveness properties. To prove liveness properties, one has to come up with a well-founded domain and a ranking function from states into the well-founded domain. The ranking function has to be such that no state leads into a higher ranked state, and, because of fairness, every state eventually must lead into a lower ranked state. Thus, we need to extend the method of invisible invariants to generate well founded domains and ranking, as well as to have the counter-part of Lemma 1 to produce cutoff values for the case of liveness properties.

Acknowledgment We wish to express our deep gratitude to Elad Shahar who, not only constructed the TLV programmable model-checker which we used to implement all the verification tasks described in this work, but also modified and extended it as we went along, graciously responding to any new needs and requests raised during the research.

References

[ABJN99] P.A. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling global conditions in parametrized system verification. In *CAV'99, LNCS 1633*, pp. 134–145, 1999.

[AK86] K. R. Apt and D. Kozen. Limits for automatic program verification of finite-state concurrent systems. *Information Processing Letters*, 22(6), 1986.

[BCG86] M.C. Browne, E.M. Clarke, and O. Grumberg. Reasoning about networks with many finite state processes. In *PODC'86*, pp. 240–248, 1986.

[BD94] J. R. Burch and D. L. Dill. Automatic verification of pipelined microprocessor control. In *CAV'94, LNCS 818*, pp. 68–80, 1994.

[CEFJ96] E.M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1/2), 1996.

[CGJ95] E.M. Clarke, O. Grumberg, and S. Jha. Verifying parametrized networks using abstraction and regular languages. In *CONCUR'95*, pp. 395–407.

[CR99a] S.J. Creese and A.W. Roscoe. Formal verification of arbitrary network topologies. In *Proc. of the Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*. CSREA Press, 1999.

[CR99b] S.J. Creese and A.W. Roscoe. Verifying an infinite family of inductions simultaneously using data independence and fdr. In *FORTE/PSTV'99*.

[CR00] S.J. Creese and A.W. Roscoe. Data independent induction over structured networks. In *Proc. of the Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'00)*. CSREA Press, 2000.

[Del00] G. Delzanno. Automatic verification of parametrized cache coherence protocols. In *CAV'00, LNCS 1855*, pp. 53–68, 2000.

[EK00] E.A. Emerson and V. Kahlon. Reducing model checking of the many to the few. In *CADE'00*, pp. 236–255, 2000.

[EN95] E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In *POPL'95*.

[EN96] E.A. Emerson and K.S. Namjoshi. Automatic verification of parameterized synchronous systems. In *CAV'96, LNCS 1102*, 1996.

[ES96] E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2), 1996.

[ES97] E. A. Emerson and A. P. Sistla. Utilizing symmetry when model checking under fairness assumptions. *TOPLAS*, 19(4), 1997.

[Ger00] S. German. Personal Communication, 2000.

[GS92] S.M. German and A.P. Sistla. Reasoning about systems with many processes. *JACM*, 39:675–735, 1992.

[GS97] V. Gyuris and A. P. Sistla. On-the-fly model checking under fairness that exploits symmetry. In *CAV'97, LNCS 1254*, 1997.

[GZ98] E.P. Gribomont and G. Zenner. Automated verification of szymanski's algorithm. In *TACAS'98, LNCS 1384*, pp. 424–438, 1998.

[HLR92] N. Halbwachs, F. Lagnier, and C. Ratel. An experience in proving regular networks of processes by modular model checking. *Acta Informatica*, 29(6/7):523–543, 1992.

[ID96] C.N. Ip and D. Dill. Verifying systems with replicated components in Murφ. In *CAV'96, LNCS 1102*, 1996.

[JL98] E. Jensen and N.A. Lynch. A proof of burn's n-process mutual exclusion algorithm using abstraction. In *TACAS'98, LNCS 1384*, pp. 409–423, 1998.

[JN00] B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In *TACAS'00, LNCS 1785*, 2000.

[KM95] R.P. Kurshan and K.L. McMillan. A structural induction theorem for processes. *Inf. and Comp.*, 117:1–11, 1995.

[KMM+97] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In *CAV'97, LNCS 1254*, pp. 424–435, 1997.

[KP00] Y. Kesten and A. Pnueli. Control and data abstractions: The cornerstones of practical formal verification. *STTT*, 4(2):328–342, 2000.

[LHR97] D. Lesens, N. Halbwachs, and P. Raymond. Automatic verification of parameterized linear networks of processes. In *POPL'97*, 1997.

[LS97] D. Lesens and H. Saidi. Automatic verification of parameterized networks of processes by abstraction. In *INFINITY'97*, 1997.

[MAB+94] Z. Manna, A. Anuchitanukul, N. Bjørner, A. Browne, E. Chang, M. Colón, L. De Alfaro, H. Devarajan, H. Sipma, and T.E. Uribe. STeP: The Stanford Temporal Prover. Technical Report STAN-CS-TR-94-1518, Stanford University, 1994.

[McM98a] K.L McMillan. Getting started with smv. Technical report, Cadence Berkeley Labs, 1998.

[McM98b] K.L. McMillan. Verification of an implementation of Tomasulo's algorithm by compositional model checking. In *CAV'98, LNCS 1427*, pp. 110–121.

[MP95a] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, New York, 1995.

[PP84] M.S. Papamarcos and J.H. Patel. A low-overhead coherence solution for multiprocessors with private cache memories. In *Proc. Int. Symp. on Shared Memory Multiprocesors (ISCA'84)*, pp. 348–354, 1984.

[PRZ01] A. Pnueli, S. Ruah, and L. Zuck. Automatic deductive verification with invisible invariants. In *TACAS'01, LNCS*, pp. 82–97, 2001.

[PS00] A. Pnueli and E. Shahar. Liveness and acceleration in parameterized verification. In *CAV'00, LNCS 1855*, pp. 328–343, 2000.

[SG89] Z. Shtadler and O. Grumberg. Network grammars, communication behaviors and automatic verification. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pp. 151–165, 1989.

[SOR93] N. Shankar, S. Owre, and J.M. Rushby. The PVS proof checker: A reference manual (draft). Technical report, Comp. Sci.,Laboratory, SRI International, 1993.

[Szy88] B. K. Szymanski. A simple solution to Lamport's concurrent programming problem with linear wait. In *Proc. 1988 International Conference on Supercomputing Systems*, pp. 621–626, 1988.

[WL89] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pp. 68–80, 1989.