Automatic Deductive Verification with Invisible Invariants*

Amir Pnueli[1], Sitvanit Ruah[1], and Lenore Zuck[2]

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

Abstract. The paper presents a method for the automatic verification of a certain class of parameterized systems. These are *bounded-data* systems consisting of N processes (N being the parameter), where each process is finite-state. First, we show that if we use the standard deductive INV rule for proving invariance properties, then all the generated verification conditions can be automatically resolved by finite-state (BDD-based) methods with no need for interactive theorem proving.
Next, 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. Combining this automatic computation of invariants with the previously mentioned resolution of the VCs (verification conditions) yields a (necessarily) incomplete but fully automatic sound method for verifying bounded-data 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".
We illustrate the method on a non-trivial example of a cache protocol, provided by Steve German.

1 Introduction

Automatic verification of infinite state systems in general, and parameterized systems in particular, have been the focus of much research recently (see, e.g., [ES96,ES97,CFJ96,GS97,ID96,LS97,RKR+00].) Most of this research concentrates on *model checking* techniques for verification of such systems, using symmetry reduction and similar methods to make model checking more tractable.

In this paper we present a method for the automatic verification of a certain class of parameterized systems using a deductive approach. The parameterized systems we study are *bounded-data* systems consisting of N processes (N being the parameter), where each process is finite-state and the number of its states is independent of N. We first show that for a large and interesting set of assertions, called *R-assertions*, there is a number, N_0, such that the verification condition

* 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.

T. Margaria and W. Yi (Eds.): TACAS 2001, LNCS 2031, pp. 82–97, 2001.

claiming that an R-assertion φ is preserved by any step of the system is valid for every $N > 1$ iff it is valid for every $N \leq N_0$. Thus, to check for validity of such verification conditions, it suffices to consider only parameterized systems with up to N_0 processes. The number N_0 is small. In fact, it is linear in the number of the local state variables of an individual process (i.e. logarithmic in the number of local states of a single process).

Using the standard deductive INV rule for proving invariance properties, all the generated verification conditions for the systems we are considering are R-assertions. Thus, for these systems, verification of invariance properties using INV can be automatically resolved by finite-state (BDD-based) methods, with no need for interactive theorem proving.

We also show how to use model-checking techniques over finite (N_0-process) instances of the parameterized system in order to derive candidates for invariant assertions. The combination of this automatic computation of invariants with the previously mentioned resolution of the verification conditions (VCs) yields a (necessarily) incomplete but fully automatic sound method for verifying bounded-data 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".

We illustrate the method on a non-trivial example of a cache protocol, provided by Steve German. In this example, N client processes may request shared or exclusive access to a shared cache line. A *Home* process coordinates the cache access. Using our approach, we managed to automatically verify the property of *coherence* by which, if one process has an exclusive access to the cache line, then no other process may have any access right to the same line, even a shared one. We verified this property for any $N > 1$ using only the instance of $N = 4$.

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 [SG92] which assumes a parameterized system where processes communicate synchronously, and shows how to verify single-index properties. Similarly, Emerson and Namjoshi [EN96] proved a PSPACE complete algorithm for verification of synchronously communicating processes. 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 Section 6 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,MP90] 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,CFJ96,GS97] study symmetry reduction in order to deal with state explosion. The work in [ID96] detects symmetries by inspection of the system description. Perhaps the closest in spirit to our work is the work of McMillan on compositional model-checking (e.g. [McM98]), which combines automatic abstraction with finite-instantiation due to symmetry. What started our research was the observation that, compared to fully deductive verification, McMillan's method requires significantly fewer auxiliary invariants, usually down to 2 auxiliary lemmas. Our explanation for this phenomenon was that, by performing model-checking instead of the usual one-step induction, his model-checker computes many of the necessary auxiliary invariants automatically. This led us to the conjecture that we can compute the full invariant characterizing the reachable states automatically by considering just a few processes, and then abstract and generalize it automatically to any number of processes, which is the basis for our method.

2 Bounded-Data Parameterized Systems

We consider systems whose variables can be declared as follows:

$$V = \begin{cases} N & : \textbf{natural where } N > 1 \\ x_1, \ldots, x_a & : \textbf{boolean} \\ y_1, \ldots, y_b & : [1..N] \\ z_1, \ldots, z_c & : \textbf{array } [1..N] \textbf{ of boolean} \end{cases}$$

Variable N is the system's parameter which, with no loss of generality, we assume to be bigger than 1. Note that we do not allow parameterized arrays whose elements range over $[1..N]$. Such data types will take us beyond the scope of bounded-data parameterized systems. We can easily extend the data-type restrictions to allow arbitrary finite types instead of just booleans. Thus, we could allow an z_r to be a parameterized array of any finite type, and let a x_r range over such a type.

We refer to the set of variables $\{y_1, \ldots, y_b\}$ as Y. In addition to the system variables, we also use a set of auxiliary variables $Aux = \{i, j, h, t, u \ldots : [1..N]\}$. We refer to the variables in $Y \cup Aux$, that range over the parametric domain $[1..N]$, as *Par-variables*. We define a class of assertions, to which we refer as R-assertions, as follows:

- x_s, $z_r[h]$, and $h = t$ are R-assertions, for $s = 1, \ldots, a$, every *Par*-variables h and t, and $r = 1, \ldots, c$. For the extended case that z_r is an array over the finite domain D_r, we also allow the atomic assertion $z_r[h] = d$ for every constant $d \in D_r$.
- If p and q are R-assertions, then so are $\neg p$, $p \vee q$, and $\exists h : p$, for every $h \in Aux$.

The other boolean operations and universal quantification can be defined using the existing operators and negation. We write $p(h)$, $q(h,t)$, to denote that the only auxiliary variables to which p (respectively q) may refer are h (respectively h, t). An R-assertion p is said to be *closed* if it contains no free occurrence of an auxiliary variable.

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*. An R-assertion characterizing the initial states.
- $\rho(V, V')$ – The *transition relation*. An R-assertion, relating the values V of the variables in state $s \in \Sigma$ to the values V' in an S-successor state $s' \in \Sigma$.

We require that ρ has the special form

$$\rho = \exists h : \bigvee_{\ell=1,\ldots,M} p_\ell(h) \ \wedge \ \forall t : q_\ell(h,t),$$

where $h, t \in Aux$, and $p_\ell(h)$, $q_\ell(h,t)$ are quantifier-free R-assertions which may refer to both V and V'.

Typically, a bounded-data parameterized system is a parallel composition $H \parallel P[1] \parallel \cdots \parallel P[N]$. The R-assertion $p_\ell(h)$ often describes the local effect of taking a transition τ_ℓ within process $P[h]$, while $q_\ell(h,t)$ describes the effect of this transition on all other processes. Usually, $q_\ell(h,t)$ will say that the local variables of all processes $P[t]$, for $t \neq h$, are preserved under a step of process $P[h]$. Note that a state s of a bounded-data system should also interpret the parameter N. We refer to $s[N]$ as the *size* of the global state s.

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.

To illustrate the representation of a parameterized system as a BDS, consider program MUX-SEM, presented in Fig. 1. The semaphore instructions "**request** x" and "**release** x" appearing in the program stand, respectively, for

$\langle \textbf{when } x = 1 \textbf{ do } x := 0 \rangle$ and $x := 1$

$$
\begin{array}{l}
\textbf{in} \quad N : \textbf{natural where } N > 1 \\
\textbf{local } x \ : \textbf{boolean where } x = 1 \\
\qquad \Big\| _{h=1}^{N} P[h] :: \left[\begin{array}{l} \textbf{loop forever do} \\ \quad \left[\begin{array}{l} I : \textbf{Non-Critical} \\ T : \textbf{request } x \\ C : \textbf{Critical} \\ E : \textbf{release } x \end{array} \right] \end{array} \right]
\end{array}
$$

Fig. 1. Program MUX-SEM

In Fig. 2, we present the BDS which corresponds to program MUX-SEM. Note that the BDS standardly contains the additional system array variable $\pi[1..N]$, which represents the program counter in each of the processes.

$$
\begin{array}{l}
V : \left\{ \begin{array}{l} N : \textbf{natural where } N > 1 \\ x : \ \textbf{boolean where } x = 1 \\ \pi : \ \textbf{array } [1..N] \textbf{ of } \{I, T, C, E\} \end{array} \right. \\
\Theta : x \ \wedge \ \forall h : [1..N] : \pi[h] = I \\
\rho : \ \exists h : [1..N] \left(\begin{array}{l} \quad \pi'[h] = \pi[h] \ \wedge \ x' = x \\ \vee \ \pi[h] = I \ \wedge \ \pi'[h] = T \ \wedge \ x' = x \\ \vee \ \pi[h] = T \wedge x = 1 \wedge \pi'[h] = C \wedge x' = 0 \\ \vee \ \pi[h] = C \ \wedge \ \pi'[h] = E \ \wedge \ x' = x \\ \vee \ \pi[h] = E \ \wedge \ \pi'[h] = I \ \wedge \ x' = 1 \end{array} \right) \ \wedge \ \forall t \neq h : \pi'[t] = \pi[t]
\end{array}
$$

Fig. 2. The BDS corresponding to program MUX-SEM.

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 $s_{\ell+1}[v]$.

The definitions of R-assertions and BDS are such that the only tests applied to *Par*-variables are equalities (and disequalities). Consequently, states, computations, and satisfaction of R-assertions are all *symmetric* with respect to an arbitrary permutation of indices. Consider the system instance $S(N_0)$, i.e., an instance of the system in which N has the value N_0. Let $\Pi : [1..N_0] \to [1..N_0]$ be a permutation on the indices $[1..N_0]$. We say that the state $\widetilde{s}$ is a Π-variant of s, denoted $\widetilde{s} = s[\Pi]$ if the following holds:

- $\widetilde{x}_r = x_r$, for every $r \in [1..a]$.
- $\widetilde{y}_r = \Pi^{-1}(y_r)$, for every $r \in [1..b]$.
- $\widetilde{z}_r[h] = z_r[\Pi(h)]$, for every $r \in [1..c]$, $h \in [1..N_0]$.

where, we write $\widetilde{v}$ to denote the value of $v \in V$ in $\widetilde{s}$, while writing simply v denotes the value of this variable in state s.

For example, applying the permutation

$$\Pi : 1 \to 2,\ 2 \to 3,\ 3 \to 1$$

to the state

$$s : \langle z[1] : 10,\ z[2] : 20,\ z[3] : 30;\ y_1 : 1;\ y_2 : 2 \rangle$$

yields the state

$$\widetilde{s} : \langle z[1] : 20,\ z[2] : 30,\ z[3] : 10;\ y_1 : 3;\ y_2 : 1 \rangle$$

Given an infinite state sequence, $\sigma : s_0, s_1, \ldots$ and a permutation Π, we define the Π-variant of σ, denoted $\sigma[\Pi]$ to be the state sequence $\sigma[\Pi] = s_0[\Pi], s_1[\Pi], \ldots$. The following claim makes the statement of symmetry precise.

Claim (Statement of Symmetry). Let $S = \langle V, \Theta, \rho \rangle$ be a BDS, and Π be a permutation with finite domain. Then

- For a closed R-assertion p and a state s, $s \models p$ iff $s[\Pi] \models p$. This leads to the following consequences:
- State $s \models \Theta$ is iff $s[\Pi] \models \Theta$.
- State s_2 is a ρ-successor of s_1 iff $s_2[\Pi]$ is a ρ-successor of $s_1[\Pi]$.
- $\sigma : s_0, s_1, \ldots$ is a computation of S iff $\sigma[\Pi]$ is a computation of S.

From now on, we will refer to R-assertions simply as assertions.

3 Verification Methods

In this section we will briefly survey the two main approaches to verification: Enumeration and Deduction. Both establish a property of the type $S \models \Box p$ for an assertion p.

3.1 The Method of Enumeration: Model Checking

For an assertion $p = p(V)$ and transition relation $\rho = \rho(V, V')$, we define the *ρ-postcondition* of p, denoted by $p \diamond \rho$, by the formula

$$p \diamond \rho = \mathit{unprime}(\exists V : p(V)\ \wedge\ \rho(V, V'))$$

The operation *unprime* is the syntactic replacement of each primed occurrence v' by its unprimed version v.

We can also define the iterated computation of postconditions:

$$p \diamond \rho^* = p\ \vee\ p \diamond \rho\ \vee\ (p \diamond \rho) \diamond \rho\ \vee\ ((p \diamond \rho) \diamond \rho) \diamond \rho\ \vee\ \cdots,$$

which, for finite-state systems, is guaranteed to terminate. Using this concise notation, verification by model checking can be summarized by the following claim:

Claim (Model Checking). Let $S = \langle V, \Theta, \rho \rangle$ be a finite-state system and p an assertion. Then, $S \models \Box p$ iff the implication

$$\Theta \diamond \rho^*\ \to\ p$$

is valid.

3.2 Deductive Verification: The Invariance Rule

Assume that we wish to prove that assertion p is an invariant of system S. The method of deductive verification suggests that the user comes up with an auxiliary assertion φ, intended to be an over-approximation of the set of reachable states, and then show that φ implies p. This can be summarized by rule INV, presented in Fig. 3.

$$\begin{array}{ll} \text{I1.} & \Theta \rightarrow \varphi \\ \text{I2.} & \varphi \wedge \rho \rightarrow \varphi' \\ \text{I3.} & \varphi \rightarrow p \\ \hline & \Box p \end{array}$$

Fig. 3. The invariance Rule INV.

An assertion φ satisfying premises I1 and I2 is called *inductive*. An inductive assertion is always an over-approximation of the set of reachable states. Premise I3 ensures that assertion φ is a *strengthening* (under-approximation) of the property p. In rare cases, the original assertion p is already inductive. In all other cases, the deductive verifier has to perform the following tasks:

T1. Divine (invent) the auxiliary assertion φ.
T2. Establish the validity of premises I1–I3.

For the case that the system S is finite-state all the assertions can be represented by BDD's. Validity of these premises can then be checked by computing the BDD of their negations, and checking that it equals 0 (false). For the case that S is not a finite-state system, for example, if it is a BDS, one traditionally uses interactive theorem provers such as PVS [SOR93] and STeP [MAB+94].

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. Therefore, our first efforts were directed towards the development of methods which will enable establishing the validity of the premises of Rule INV for bounded-data parameterized systems in a fully automated manner.

4 Deciding the Verification Conditions

In this section, we outline a decision procedure for establishing the validity of the verification conditions generated by rule INV for bounded-data parameterized systems. Consider first the case that the auxiliary assertion φ has the form $\varphi = \forall i : \psi(i)$, where $\psi(i)$ is a quantifier-free (R-)assertion. The most complex verification condition is premise I2 which can be written as:

$$(\forall j : \psi(j)) \ \wedge \ (\exists h : \bigvee_{\ell=1,\ldots,M} p_\ell(h) \ \wedge \ \forall t : q_\ell(h,t)) \quad \rightarrow \quad \forall i : \psi'(i) \qquad (1)$$

The following claim states that, for a bounded-data parameterized system $S(N)$, condition (1) can be decided by establishing it over finitely (and not too) many instances of $S(N)$.

Claim. Let $S(N)$ be a bounded-data parameterized system. Then, the implication (1) is valid over $S(N)$ for all $N > 1$ iff it is valid over $S(N)$ for all N, $1 < N \leq 2b + 2$, where b is the size of Y.

For example, the claim states that it is sufficient to check the premises of rule INV over MUX-SEM(2) in order to establish their validity over all instances of MUX-SEM(N).

Proof: (Sketch) Let $N_0 = 2 + 2b$. To prove the claim, it is sufficient to show that the negation of condition (1), given by

$$(\forall j : \psi(j)) \ \wedge \ (\exists h : \bigvee_{\ell=1,\ldots,M} p_\ell(h) \ \wedge \ \forall t : q_\ell(h,t)) \ \wedge \ \exists i : \neg\psi'(i) \qquad (2)$$

is satisfiable for some $N > 1$ iff it is satisfiable for some $1 < N \leq N_0$. Clearly, formula (2) is satisfiable iff the formula

$$(\forall j : \psi(j)) \ \wedge \bigvee_{\ell=1,\ldots,M} (p_\ell(h) \ \wedge \ \forall t : q_\ell(h,t)) \ \wedge \ \neg\psi'(i) \qquad (3)$$

is satisfiable. It suffices to show that if formula (3) is satisfiable over a state (pair) of size $N > N_0$, it is also satisfiable over a state (pair) of size N_0.

Let s be a state of size $N_1 > N_0$ which satisfies assertion (3). The states s assigns to the variables $V_{aug} = \{h, i, y_1, y'_1, \ldots, y_b, y'_b\}$ values in the domain $[1..N_1]$. Let $\alpha \leq N_0$ be the number of the *different* values assigned to those variables, and assume these values are $v_1 < v_2 < \cdots < v_\alpha$. There obviously exists a permutation Π on $[1..N_1]$ such that $\Pi^{-1}[v_k] = k$ for every $k = 1, \ldots, \alpha$. Let $\widetilde{s}$ be the Π-variant of s, applying the permutation-induced transformation described in Section 2 to the augmented set of state variables $V_{aug} = V \cup \{h, i\}$. The size of $\widetilde{s}$ is N_1, and, according to Claim 2, it satisfies assertion (3), which is a closed assertion relative to the augmented variable set $V \cup \{h, i\}$.

We proceed to show how to derive a new state $\widehat{s}$ of size $\alpha \leq N_0$ which also satisfies assertion (3). The state $\widehat{s}$ is defined by letting $\widetilde{s}[N] = \alpha$ and letting $\widehat{s}$ and $\widetilde{s}$ agree on the interpretation of the variables in $h, i, y_1, y'_1, \ldots, y_b, y'_b, x_1, x'_1, \ldots, x_a, x'_a$. For the remaining variables (the z_r's), we let $\widehat{s}$ and $\widetilde{s}$ agree on the interpretation of every variable $z_r[k]$ and $z'_r[k]$ where $r \in [1..c]$ and $k \leq \alpha$.

It remains to show that if $\widetilde{s}$ satisfies the N_1-version of assertion (3) then $\widehat{s}$ satisfies the α-version of assertion (3), where the assertions are

$$(\bigwedge_{j=1}^{N_1} \psi(j)) \ \wedge \bigvee_{\ell=1,\ldots,M} (p_\ell(h) \ \wedge \bigwedge_{t=1}^{N_1} q_\ell(h,t)) \ \wedge \ \neg\psi'(i), \qquad (4)$$

and

$$(\bigwedge_{j=1}^{\alpha} \psi(j)) \ \wedge \bigvee_{\ell=1,\ldots,M} (p_\ell(h) \ \wedge \bigwedge_{t=1}^{\alpha} q_\ell(h,t)) \ \wedge \ \neg\psi'(i) \qquad (5)$$

respectively.

Since the difference between the two assertions is that the conjunctions in assertion (5) extend only over the $[1..\alpha]$ subrange of the conjunctions in assertion

(4), and since $\widehat{s}$ and $\widetilde{s}$ agree on the interpretation of variables in this subrange, we conclude that $\widehat{s}$ satisfies assertion (5). ∎

Claim 4 can be extended in several different ways. For example, we can trivially modify it to establish that premises I1 and I3 of Rule INV can also be checked only for systems of size not exceeding $2b+2$. Another useful modification applies to the case of *Par*-deterministic systems. A bounded-data system is said to be *Par-deterministic* if, for every *Par*-variable y_r and every disjunct $p_\ell(h)$ of the transition relation, $p_\ell(h)$ contains a conjunct of the form $y'_r = u$ for some unprimed *Par*-variable u. Recall that the bound of $2b+2$ was derived in order to cover the possibility that $h, i, y_1, y'_1, \ldots, y_b, y'_b$ may all assume disjoint values. Under a *Par*-deterministic transition relation, all the primed variables must assume values that are equal to the values of some unprimed variables. Therefore, the set of variables $h, i, y_1, y'_1, \ldots, y_b, y'_b$ can assume at most $b+2$ distinct values. This leads to the following corollary:

Corollary 1. *Let $S(N)$ be a Par-deterministic BDS. 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, $1 < N \leq b+2$.*

The last extension considers the case that both the property p to be proven and the auxiliary invariant φ have the form $\forall h, t : \psi(h, t)$ for some quantifier-free (R-)assertion ψ.

Corollary 2. *Let $S(N)$ be a bounded-data parameterized system, and let p and φ both have the form $\forall h, t : \psi(h, t)$. 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, $1 < N \leq 2b+3$. In the case that $S(N)$ is Par-deterministic, it is sufficient to check the premises for $N \leq b+3$.*

5 Automatic Calculation of the Auxiliary Invariants

Providing a decision procedure for the premises of rule INV greatly simplifies the process of deductive verification. Yet, it still leaves open the task of inventing the strengthening assertion φ. As illustrated in the next section, this strengthening assertion may become quite complex for all but the simplest systems.

Here we propose a heuristic for an algorithmic construction of an inductive assertion for a given bounded-data parameterized system. Let us consider first the case that we are looking for an inductive assertion of the form $\varphi = \forall h : \psi(h)$. The construction algorithm can be described as follows:

Algorithm 1. *Compute Auxiliary Assertion of the form $\forall h : \psi(h)$*

1. Let *reach* be the assertion characterizing all the reachable states of system $S(N_0)$, where $N_0 = 2b+2$ (or $b+2$ if S is *Par*-deterministic). Since $S(N_0)$ is finite-state, *reach* can be computed by letting $\mathit{reach} := \Theta \diamond \rho^*$.
2. Let ψ_1 be the assertion obtained from *reach* by projecting away all the references to variables subscripted by indices other than 1. Technically, this is done by using BDD operations for computing
$$\psi_1 = \exists z_1[2], \ldots, z_1[N_0], \ldots, z_c[2], \ldots, z_c[N_0] : \mathit{reach}$$

3. Let $\psi(h)$ be the assertion obtained from ψ_1 by *abstraction*, which involves the following transformations:
   - Replace any reference to $z_r[1]$ by a reference to $z_r[h]$.
   - Replace any sub-formula of the form $y_r = 1$ by the formula $y_r = h$, and any sub-formula of the form $y_r = v$ for $v \neq 1$ by the formula $y_r \neq h$.

Let us illustrate the application of this algorithm to program MUX-SEM (as presented in Fig. 1). Since, for this program, $b = 0$, we take $N_0 = 2$ and obtain

$$\begin{array}{ll} \mathit{reach}: & \left(\begin{array}{l} (x=1) \wedge \pi[1] \in \{I,T\} \wedge \pi[2] \in \{I,T\} \\ \vee\ (x=0) \wedge (\pi[1] \in \{I,T\} \leftrightarrow \pi[2] \notin \{I,T\}) \end{array}\right) \\ \psi_1: & (x=1) \rightarrow \pi[1] \in \{I,T\} \\ \psi(h) & (x=1) \rightarrow \pi[h] \in \{I,T\} \end{array}$$

Unfortunately, when we take the proposed assertion $\varphi : \forall h : (x = 1) \rightarrow \pi[h] \in \{I, T\}$ we find out that it is not inductive over $S(2)$. This illustrates the fact that the above algorithm is not guaranteed to produce inductive assertions in all cases.

Another version of the algorithm can be used to compute candidates for inductive assertions of the form $\varphi : \forall h \neq t : \psi(h, t)$.

Algorithm 2. *Compute Auxiliary Assertion of the form* $\forall h \neq t : \psi(h, t)$

1. Let *reach* be the assertion characterizing all the reachable states of system $S(N_0)$, where $N_0 = 2b + 3$ (or $b + 3$ if S is *Par*-deterministic).
2. Let $\psi_{1,2}$ be the assertion obtained from *reach* by projecting away all the references to variables subscripted by indices other than 1 or 2.
3. Let $\psi(h, t)$ be the assertion obtained from $\psi_{1,2}$ by *abstraction*, which involves the following transformations:
   - Replace any reference to $z_r[1]$ by a reference to $z_r[h]$ and any reference to $z_r[2]$ by a reference to $z_r[t]$.
   - Replace any sub-formula of the form $y_r = 1$ by the formula $y_r = h$, any sub-formula of the form $y_r = 2$ by the formula $y_r = t$, and any sub-formula of the form $y_r = v$ for $v \notin \{1, 2\}$ by the formula $y_r \neq h \wedge y_r \neq t$.

Let us apply this algorithm again to system MUX-SEM. This time, we take $N_0 = 3$ and compute:

$$\begin{array}{ll} \mathit{reach}: & (\pi[1] \in \{C,E\}) + (\pi[2] \in \{C,E\}) + (\pi[3] \in \{C,E\}) + y = 1 \\ \psi_{1,2}: & \left(\begin{array}{l} \pi[1] \in \{C,E\} \rightarrow (x=0) \ \wedge\ \pi[2] \in \{I,T\} \\ \wedge\ \pi[2] \in \{C,E\} \rightarrow (x=0) \ \wedge\ \pi[1] \in \{I,T\} \end{array}\right) \\ \psi(h,t): & \left(\begin{array}{l} \pi[h] \in \{C,E\} \rightarrow (x=0) \ \wedge\ \pi[t] \in \{I,T\} \\ \wedge\ \pi[t] \in \{C,E\} \ \rightarrow (x=0) \ \wedge\ \pi[h] \in \{I,T\} \end{array}\right) \end{array}$$

Taking $\varphi = \forall h \neq t : \psi(h, t)$ yields an assertion which is inductive over $S(3)$. By Corollary (2), it follows that φ is inductive for all $S(N)$. It is straightforward to check that φ implies the property of mutual exclusion $\forall h \neq t : \neg(\pi[h] = C \ \wedge\ \pi[t] = C)$ which we wished to establish for program MUX-SEM.

5.1 The Integrated Processes

The description of the two algorithms for computing auxiliary assertions may have given some of the readers the false impression that there is a manual step involved. For example, that after computing ψ_1 in Algorithm (1), we print it out and ask the user to perform the abstraction herself. This is certainly not the case. The whole process of deriving the candidate for inductive auxiliary assertion and utilizing it for an attempt to verify the desired property is performed in a fully automated manner. In fact, without an explicit request, the user never sees the generated candidate assertion, which is the reason we refer to this method as "verification by invisible invariants".

To explain how the entire process is performed, we observe that steps (2) and (3) of Algorithm (1) obtain a symbolic representation of $\psi(h)$. However, to check that it is inductive over $S(N_0)$, we immediately instantiate h in $\psi(h)$ to form $\bigwedge_{j=1}^{N_0} \psi(j)$. In the integrated process, we perform these three steps together. This is done by defining an *abstraction relation* α_j for each $j \in [1..N_0]$. The abstraction relation is given by

$$\alpha_j : \quad \bigwedge_{r=1}^{a} (x'_r = x_r) \ \wedge \ \bigwedge_{r=1}^{b} \Big((y'_r = j) \leftrightarrow (y_r = 1) \Big) \ \wedge \ \bigwedge_{r=1}^{c} (z'_r[j] = z_r[1])$$

This relation defines an abstract state consisting of the interpretation of a primed copy V' which only cares about the interpretation of $z'_r[j]$, whether y'_r equals or is unequal to j, and the precise values of x_r. These values correspond to the interpretation of these variables for $j = 1$ in the unprimed state. Given the assertion *reach* which characterizes all the reachable states of $S(N_0)$, we can form the assertion $\psi_j = \mathit{reach} \diamond \alpha_j$. Then we claim that

The state $\widetilde{s}$ is in $\|\psi_j\|$ iff there exists a state $s \in \|\mathit{reach}\|$, such that

$$\begin{aligned} \widetilde{x}_r &= x_r \qquad \text{for every } r \in [1..a] \\ \widetilde{y}_r = j \text{ iff } y_r &= 1 \text{ for every } r \in [1..b] \\ \widetilde{z}_r[j] &= z_r[1] \quad \text{for every } r \in [1..c] \end{aligned}$$

Thus, we reuse the operator $\diamond$ for performing abstraction+instantiation instead of computation of a successor, which is its customary use.

With this notation, we can describe the full verification process as follows:

Verification Process 3. Verify property p, using a singly indexed auxiliary assertion.

1. Let $\mathit{reach} := \Theta \diamond \rho^*$, computed over $S(N_0)$ for an appropriately chosen N_0.
2. Let $\psi_j := \mathit{reach} \diamond \alpha_j$, for each $j \in [1..N_0]$.
3. Let $\varphi := \bigwedge_{j=1}^{N_0} \psi_j$.
4. Check that φ is inductive over $S(N_0)$.
5. Check that $\varphi \ \rightarrow \ p$ is valid.

If tests (4) and (5) both yield positive results, then property p has been verified.

To illustrate the application of Verification Process (3), consider the augmented version of program MUX-SEM, presented in Fig. 4. In this program, we added an auxiliary variable *last_entered* which is set to h whenever process $P[h]$ enters its critical section. Applying Verification Process (3) to this program, we obtained the calculated invariant

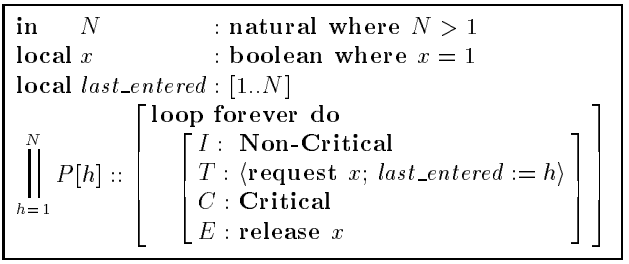

Fig. 4. Augmented Program MUX-SEM

$$\varphi : \quad \forall h : \pi[h] \in \{C, E\} \;\leftrightarrow\; (x = 0 \;\wedge\; last\_entered = h)$$

The candidate assertion φ is inductive and also implies the property of mutual exclusion, specifiable as

$$p : \quad \forall h \neq t : \neg(\pi[h] = C \;\wedge\; \pi[t] = C)$$

To handle the case of an auxiliary assertion which depends on two different indices, we define the abstraction relations

$$\alpha_{ht} : \quad \left(\begin{array}{l} \bigwedge_{r=1}^{a} (x'_r = x_r) \\ \wedge \bigwedge_{r=1}^{b} \Big((y'_r = h) \leftrightarrow (y_r = 1) \Big) \;\wedge\; \Big((y'_r = t) \leftrightarrow (y_r = 2) \Big) \\ \wedge \bigwedge_{r=1}^{c} (z'_r[h] = z_r[1]) \;\wedge\; (z'_r[t] = z_r[2]) \end{array} \right)$$

We then formulate the verification process for doubly indexed assertions:
Verification Process 4. Verify property p, using a doubly indexed auxiliary assertion.

1. Let $reach := \Theta \diamond \rho^*$, computed over $S(N_0)$ for an appropriately chosen N_0.
2. Let $\psi_{ht} := reach \diamond \alpha_{ht}$, for each $h < t \in [1..N_0]$.
3. Let $\varphi := \bigwedge_{h<t\in[1..N_0]} \psi_{ht}$.
4. Check that φ is inductive over $S(N_0)$.
5. Check that $\varphi \;\rightarrow\; p$ is valid.

6 German's Cache Case Study

In this section we illustrate the application of the invisible-invariants verification method to a case study which is a simple cache algorithm provided to us by Steve German [Ger00]. The algorithm consists of a central controller called *Home* and N client processes $P[1], \ldots, P[N]$. Each of the clients communicates with *Home* via the following channels:

- *channel1* – Client $P[c]$ uses this channel to send *Home* requests for either *shared* or *exclusive* access to the cache line.
- *channel2* –*Home* uses this channel to send $P[c]$ permissions (grants) for the requested access rights. It also sends on this channel requests to $P[c]$ to invalidate its cache status.

in N : **natural where** $N > 1$
type $message = \{empty, req\_shared, req\_exclusive, invalidate, invalidate\_ack, grant\_shared, grant\_exclusive\}$
type $cache\_state = \{invalid, shared, exclusive\}$
local $channel1, channel2, channel3$: **array**$[1..N]$ **of message where** $\forall i : [1..N].(channel1[i] = channel2[i] = channel3[i] = empty)$
local $sharer\_list, invalidate\_list$: **array**$[1..N]$ **of bool where** $\forall i : [1..N].(sharer\_list[i] = invalidate\_list[i] = 0)$
local $exclusive\_granted$: **bool where** $exclusive\_granted = 0$
local $curr\_command$: **message where** $curr\_command = empty$
local $curr\_client$: $[1..N]$ **where** $curr\_client = 1$
local $cache$: **array**$[1..N]$ **of cache_state where** $\forall i : [1..N].(cache[i] = empty)$

Fig. 5. Variables for German's cache algorithm

- *channel3* – Client $P[c]$ uses this channel to send *Home* acknowledgments of invalidation of the client's cache status.

Fig. 5 presents the variables used in the algorithm.

The algorithm can be presented as

$$Home \quad \Big\| \quad \Big\|_{c=1}^{N} P[c]$$

An SPL program for *Home* is presented in Figure Fig. 6, and an SPL program for $P[c]$ is presented in Fig. 7.

The main property we wish to verify for this system is that of *coherence* by which there cannot be two clients, c and d, such that $P[c]$ holds an exclusive access to the cache line while $P[d]$ holds a shared access to the same cache line at the same time. This can be specified by the invariance of the assertion

$$\forall c \neq d : \neg(cache[c] = exclusive \ \wedge \ cache[d] = shared) \tag{6}$$

Following are the results of our verification experiments applied to the cache algorithm:

1. We applied Verification Process (3) to the cache program. The computed candidate assertion failed to be inductive.
2. We augmented the cache program with an auxiliary variable *last_granted* which is assigned the value of *curr_client* in transitions m_0 and m_1. We then applied Verification Process (3) to the augmented program. This time, the candidate assertion proved to be inductive and implied the property of coherence. It took 1.97 seconds to compute the candidate assertion, and 31.43 seconds to check that it is inductive (over an instance of the program with $N = 4$).
3. We applied Verification Process (4) to the original cache program. It produced an inductive assertion which implied the property of coherence. It took 15.42 seconds to compute the candidate assertion, and 186.82 seconds to check that it is inductive.

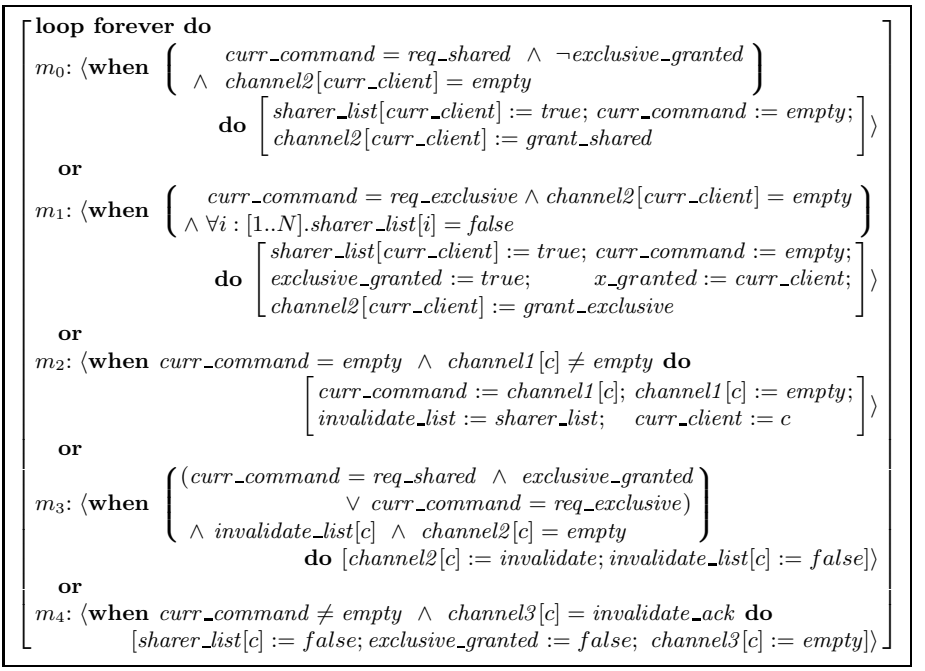

Fig. 6. Program for *Home*

We repeated these experiments over two erroneous versions of the cache program, also provided to us by Steve German. In both cases, Verification Process (4) produced inductive assertions but they failed to imply the property of coherence.

References

ABJN99. P.A. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling global conditions in parametrized system verification. In CAV'99, LNCS 1633, pages 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 *Proc. 5th ACM Symp. Princ. of Dist. Comp.*, pages 240–248, 1986.

CFJ96. E.M. Clarke, , R. Enders T. Filkron, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1/2), 8 1996. Preliminary version appeared in 5th CAV, 1993.

CGJ95. E.M. Clarke, O. Grumberg, and S. Jha. Verifying parametrized networks using abstraction and regular languages. In *6th International Conference on Concurrency Theory (CONCUR'95)*, pages 395–407, Philadelphia, PA, August 1995.

$$
\begin{array}{l}
\ell_0\colon \textbf{skip} \\
\quad \textbf{or} \\
\ell_1\colon \langle \textbf{when } \mathit{cache}[c] = \mathit{invalid} \ \wedge \ \mathit{channel1}[c] = \mathit{empty} \textbf{ do} \\
\qquad\qquad [\mathit{channel1}[c] := \mathit{req\_shared}]\rangle \\
\quad \textbf{or} \\
\ell_2\colon \langle \textbf{when } (\mathit{cache}[c] = \mathit{invalid} \ \vee \ \mathit{cache}[c] = \mathit{shared}) \wedge \mathit{channel1}[c] = \mathit{empty} \textbf{ do} \\
\qquad\qquad [\mathit{channel1}[c] := \mathit{req\_exclusive}]\rangle \\
\quad \textbf{or} \\
\ell_3\colon \langle \textbf{when } \mathit{channel2}[c] = \mathit{invalidate} \ \wedge \ \mathit{channel3}[c] = \mathit{empty} \textbf{ do} \\
\qquad [\mathit{channel2}[c] := \mathit{empty}; \mathit{channel3}[c] := \mathit{invalidate\_ack}; \mathit{cache}[c] := \mathit{invalid}]\rangle \\
\quad \textbf{or} \\
\ell_4\colon \langle \textbf{when } \mathit{channel2}[c] = \mathit{grant\_shared} \textbf{ do} \\
\qquad\qquad [\mathit{cache}[c] := \mathit{shared}; \mathit{channel2}[c] := \mathit{empty}]\rangle \\
\quad \textbf{or} \\
\ell_5\colon \langle \textbf{when } \mathit{channel2}[c] = \mathit{grant\_exclusive} \textbf{ do} \\
\qquad\qquad [\mathit{cache}[c] := \mathit{exclusive}; \mathit{channel2}[c] := \mathit{empty}]\rangle
\end{array}
$$

Fig. 7. Program for Process $P[c]$

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)*, Las Vegas, 1999. CSREA Press.

CR99b. S.J. Creese and A.W. Roscoe. Verifying an infinite family of inductions simultaneously using data independence and fdr. In *Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing and Verification (FORTE/PSTV'99)*, Beijing, 1999. Kluwer Academic Publishers.

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)*, Las Vegas, June 2000. CSREA Press.

EK00. E.A. Emerson and V. Kahlon. Reducing model checking of the many to the few. In *17th International Conference on Automated Deduction (CADE-17)*, pages 236–255, 2000.

EN95. E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In POPL'95, 1995.

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), 8 1996. Preliminary version appeared in 5th CAV, 1993.

ES97. E. A. Emerson and A. P. Sistla. Utilizing symmetry when model checking under fairness assumptions. *ACM Trans. Prog. Lang. Sys.*, 19(4), 1997. Preliminary version appeared in 7th CAV, 1995.

Ger00. S. German. Personal Communication, 2000.

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, pages 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, pages 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. *Information and Computation*, 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, pages 424–435, 1997.

KP00. Y. Kesten and A. Pnueli. Control and data abstractions: The cornerstones of practical formal verification. *Software Tools for Technology Transfer*, 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 *2nd International Workshop on the Verification of Infinite State Systems (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, Dept. of Comp. Sci., Stanford University, Stanford, California, 1994.

McM98. K.L. McMillan. Verification of an implementation of Tomasulo's algorithm by compositional model checking. In CAV'98, LNCS 1427, pages 110–121, 1998.

MP90. Z. Manna and A. Pnueli. An exercise in the verification of multi – process programs. In W.H.J. Feijen, A.J.M van Gasteren, D. Gries, and J. Misra, editors, *Beauty is Our Business*, pages 289–301. Springer-Verlag, 1990.

PS00. A. Pnueli and E. Shahar. Livenss and acceleraiton in parameterized verification. In CAV'00, LNCS 1855, 2000.

RKR+00. A. Roychoudhury, K. Narayan Kumar, C. R. Ramakrishnan, I.V. Ramakrishnan, and S.A. Smolka. Verification of parameterized systems using logic program transformations. In TACAS'00, LNCS 1785, 2000.

SG89. Z. Shtadler and O. Grumberg. Network grammars, communication behaviors and automatic verification. In CAV'89, LNCS 407, pages 151–165, 1989.

SG92. A.P. Sistla and S.M. German. Reasoning about systems with many processes. *J. ACM*, 39:675–735, 1992.

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

WL89. P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In CAV'89, LNCS 407, pages 68–80, 1989.