Invisible Safety of Distributed Protocols*

Ittai Balaban[1], Amir Pnueli[1], and Lenore D. Zuck[2]

[1] New York University, New York, {balaban,amir}@cs.nyu.edu
[2] University of Illinois at Chicago, lenore@cs.uic.edu

Abstract. The method of "Invisible Invariants" has been applied successfully to protocols that assume a "symmetric" underlying topology, be it cliques, stars, or rings. In this paper we show how the method can be applied to proving safety properties of distributed protocols running under arbitrary topologies. Many safety properties of such protocols have reachability predicates, which, on first glance, are beyond the scope of the Invisible Invariants method. To overcome this difficulty, we present a technique, called "coloring," that allows, in many instances, to replace the second order reachability predicates by first order predicates, resulting in properties that are amenable to Invisible Invariants, where "reachable" is replaced by "colored." We demonstrate our techniques on several distributed protocols, including a variant on Luby's Maximal Independent Set protocol, the Leader Election protocol used in the IEEE 1394 (Firewire) distributed bus protocol, and various distributed spanning tree algorithms. All examples have been tested using the symbolic model checker TLV.

1 Introduction

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 $N > 1$. One of the most powerful approaches to verification which is not restricted to finite-state systems is *deductive verification*. This approach is based on a set of proof rules in which the user has to establish the validity of a list of premises in order to validate a given property of the system. The two tasks that the user has to perform are:

1. Identify some auxiliary constructs which appear in the premises of the rule;
2. Use the auxiliary constructs to establish the logical validity of the premises.

When performing manual deductive verification, the first task is usually the more difficult, requiring ingenuity, expertise, and a good understanding of the behavior of the program and the techniques for formalizing these insights. The second task is often performed using theorem provers such as PVS[1] or STeP [2], which require user guidance and interaction, and place additional burden on the user. The difficulties in the execution of these two tasks are the main reason why deductive verification is not used more widely.

* This research was supported in part by NSF grant CCR-0205571 and ONR grant N00014-99-1-0131.

A representative case is the verification of invariance properties using the *invariance rule* of [3], which is described in Fig. 1. In order to prove that assertion p is an invariant of program P, the rule calls for an *auxiliary assertion* φ that is *inductive* and strengthens (implies) p. Premise I1 requires φ to hold at any initial states, which are characterized by the assertion Θ. Premise I2 requires that every ρ-successor of a φ-state is also φ-state, where ρ is the transition relation. Finally, premise I3 specifies that φ strengthens p. The main challenge in applying INV is identifying a good φ when p itself is not inductive.

$$\frac{\begin{array}{l}\text{I1. } \Theta \rightarrow \varphi \\ \text{I2. } \varphi \ \wedge \ \rho \rightarrow \varphi' \\ \text{I3. } \varphi \rightarrow p\end{array}}{p}$$

Fig. 1. The Proof Rule INV

In [4, 5] we introduced the method of *invisible invariants*, which proposes a method for automatic generation of the auxiliary assertion φ for parameterized systems, as well as an efficient algorithm for checking the validity of the premises of the invariance rule. See [6] for a tool that implements the idea.

The generation of invisible auxiliary constructs is based on the following idea: It is often the case that an auxiliary assertion φ for a parameterized system has one of the forms $q(i)$, $\forall i.q(i)$ or, more generally, $\forall i \neq j.q(i,j)$. We construct an instance of the parameterized system taking a fixed value N_0 for the parameter N. For the finite-state instantiation $S(N_0)$, we compute, using BDD-techniques, some assertion ψ, which we wish to generalize to an assertion in the required form. Let r_1 be the projection of ψ on process index 1, obtained by discarding references to all variables which are local to all processes other than $P[1]$. We take $q(i)$ to be the generalization of r_1 obtained by replacing each reference to a local variable $P[1].x$ by a reference to $P[i].x$. The obtained $q(i)$ is our candidate for the body of the inductive assertion $\varphi : \forall i.q(i)$. We refer to this part of the process as *proj-gen*. For example, when generating invariants, ψ is the set of reachable states of $S(N_0)$. The process can easily be generalized to generate assertions of the type $\forall i_1, \ldots, i_k.p(\vec{i})$.

Having obtained a candidate for the assertion φ, we still have to check the validity of the premises of the proof rule we wish to employ. Under the assumption that our assertional language is restricted to the predicates of equality and inequality between bounded range integer variables (which is adequate for many of the parameterized systems we considered), we proved a *small model* theorem, according to which, for a certain type of assertions, there exists a (small) bound N_0 such that such an assertion is valid for every N iff it is valid for all $N \leq N_0$. This enables using BDD techniques to check the validity of such an assertion. The assertions covered by the theorem are those that can be written in the form $\forall \vec{i} \exists \vec{j}.\psi(\vec{i},\vec{j})$, where $\psi(\vec{i},\vec{j})$ is a quantifier-free assertion that may refer only to the global variables and the local variables of $P[i]$ and $P[j]$, where the variables are restricted to be stratified. Thus, for example, if we have a finite domain and an index domain (that ranges over the process id's $[1..N]$), stratification requires that every array is a mapping from the index domain into the finite domain, but rules out arrays from the index domain into itself.

Being able to validate the premises on $S[N_0]$ has the additional important advantage that the user never sees the automatically generated auxiliary assertion φ. This assertion is produced as part of the procedure and is immediately consumed in order to validate the premises of the rule. Being generated by symbolic BDD techniques, the representation of the auxiliary assertions is often extremely unreadable and non-intuitive, and will usually not contribute to a better understanding of the program or its proof. Because the user never gets to see it, we refer to this method as the "method of *invisible invariants*."

As shown in [4, 5], many concurrent systems are stratified (or can be stratified), and the result of embedding a $\forall\vec{i}.q(\vec{i})$ candidate inductive invariant in the main proof rule used for their safety properties results in premises that fall under the small model theorem. In the past we have not studied protocols for general topologies, mainly because many of these require reachability analysis, which is not a first order predicate, and therefore was beyond our methods. Thus, all the systems we applied the invisible invariant method (or its successors that handle liveness), have an underlying "trivial" topology, be it a star, a clique, or a ring.

In this paper we study applications of the method of invisible invariants to arbitrary fixed topologies. We first present a small-model theorem that applies to such systems and demonstrate its application on a variant of Luby's maximal independent set protocol [7]. We then study protocols whose specifications include reachability predicates. To handle reachability with an invisible-invariant-like strategy, we augment a given protocol with a coloring scheme that starts at one node (the *initial node*), and propagates colors to adjacent non-colored nodes. At each point in the coloring, only nodes that are reachable from the initial node are colored, and when the coloring terminates, all nodes reachable from the initial node are colored. The coloring allows to replace the second-order reachability predicate with a first order *colored* predicate.

Related Work

We are not aware of any work that deals specifically with automatic verification of distributed algorithms. Most related to the work here is the work on automatic verification of parameterized systems. Our work extends the work surveyed in [8]. The PAX project (e.g., [9]) models parameterized systems in WS1S on which abstractions are computed and checked in MONA. The index predicates (e.g., [10]) combine predicate abstraction with a heuristic, similar to that used here, for constructing quantified invariants.

There have been numerous verification efforts specifically targeted at various aspects of the IEEE 1394 tree identification protocol, among them are [11, 12]. However, none of these works attempt at full automation. The work in [13] deals with the probabilistic aspect of the protocol, which we ignore in the work reported here. (We should, however, state that we have automatically verified the probabilistic aspects of the protocol using methods that are outside the scope of this paper.) For an in depth survey of previous verification efforts of the protocol see [11].

The work in [14] uses a coloring scheme, somewhat different than ours, to obtain over-approximation of reachability predicates for the purpose of shape-analysis. Since we deal with a fixed topology, our coloring scheme is precise with respect to reachability.

The paper is organized as follows: Section 2 demonstrates how we model Luby's maximal independent set and the leader election protocols. Section 3 presents the formal model of programs over arbitrary topologies, as well as a small model result. Section 4 formalizes and demonstrates use of the coloring augmentation, Section 5 summarizes runtime and verification results, and Section 6 discusses future work and concludes.

2 Examples of Distributed Protocols

To demonstrate our techniques, we present two examples of distributed protocols and their safety properties. The first is a variant of Luby's maximal independent set (MIS) protocol ([7]) and the second is the Leader Election protocol [15], which also serves as the *tree identification* protocol used in the IEEE 1394 bus specification [13].

In all of our examples, we assume a network of N processes whose id's are $[1..N]$. The interconnection among the processes is described by the boolean matrix Q, where $Q[i, j]$ denotes a direct link from i to j, and $\neg Q[i, j]$ denotes the absence of such a link. We assume that the communication between neighbors is bi-directional, therefore $Q[i, j] = Q[j, i]$ for all i and j.

2.1 Luby's MIS Protocol

The goal of the MIS protocol is to define a maximally independent set among the participating processes, i.e., a set which is independent (no two adjacent nodes are members of it) and is locally maximal (every node outside the set has a neighbor in the set). The protocols proceeds by letting processes, all of whom are initially undecided, to either enter the set ("win") or give up ("lose"). Processes that are winners or losers halt. The original protocol is synchronous, consisting of a sequence of steps, which consist of three phases: In the first phase, each process draws a number from a fixed range and sends the result to all its neighbors. In the second phase, each process that holds the maximum value among its neighbors joins the set (i.e., wins) and sends a message to that effect to all its neighbors. In the third, each process that receives a message from a neighbor that joins the set, declares itself a loser.

Since we are interested in safety properties only, and since the role of the probabilistic choices it to guarantee convergence – that every process eventually wins or loses – and the particular values used determine how fast convergence is achieved, we ignore the probabilistic aspect of the protocol and let values be non-deterministically chosen from $\{H, L\}$. Also, for technical reasons that will become clear when we prove the small model theorem, we choose to represent the protocol as asynchronous, where we impose the synchronicity required by letting each process be in one of three phases, and letting the phases of all the processes be shared variables. Finally, to avoid explicit communication rounds, we assume that the values drawn, as well as the win/lose state of each process, are shared between its neighbors.

Each process i has a variable $state[i] \in \{playing, lost, won\}$ that is initially $playing$, and a variable $\mathbf{phase}[i] \in \{0, 1, 2\}$ that is initially 0. When $\mathbf{phase}[i] = k$, the process is in the $(k+1)^{st}$ phase of the three mentioned above. The program MIS is represented in Fig. 2. Each process loops as long as $state[i] = playing$. As a first step in the loop

body, the process waits until all neighbors reach a consensus about the current phase. Such a consensus is reached if all playing neighbors have a phase which either equals to $\mathbf{phase}[i]$ or is the phase following $\mathbf{phase}[i]$. We represent this synchronization as an atomic test, allowing $P[i]$ to observe in one step the values of all of its neighbors. This assumption can be relaxed without affecting the correctness of the algorithm.

Q: **array** $[1..N]$ **of array** $[1..N]$ **of bool where** $\forall i,j.Q[i,j] = Q[j,i]$
$state$: **array** $[1..N]$ **of** $\{playing, won, lost\}$ **init** $\forall i.state[i] = playing$
val: **array** $[1..N]$ **of** $\{H, L\}$
$\mathbf{phase}$: **array** $[1..N]$ **of** $\{0,1,2\}$ **init** $\forall i.\mathbf{phase}[i] = 0$

$$\Big|\Big|_{i=1}^{N} P[i] :: \left[\begin{array}{l}\textbf{while } state[i] = playing \textbf{ do}\\ \quad\left[\begin{array}{l}\textbf{await } \forall j \neq i.Q[i,j] \wedge (state[j] = playing) \;\rightarrow\\ \qquad\qquad \mathbf{phase}[j] \in \{\mathbf{phase}[i], \mathbf{phase}[i]{+}1 \bmod 3\}\\ \textbf{if } phase[i] = 0 \textbf{ then } val[i] := \{H, L\}\\ \textbf{elsif } \mathbf{phase}[i] = 1 \;\wedge\; val[i] = H \;\wedge\; \forall j \neq i.Q[i,j] \rightarrow val[j] = L\\ \quad\textbf{then } state[i] := won\\ \textbf{elsif } \mathbf{phase}[i] = 2 \;\wedge\; \exists j \neq i.Q[i,j] \;\wedge\; state[j] = won \textbf{ then } state[i] := lost\\ \mathbf{phase}[i] := \mathbf{phase}[i]{+}1 \bmod 3\end{array}\right]\end{array}\right]$$

Fig. 2. Program MIS

The safety properties of MIS are *independence*:

$$Ind: \qquad \forall i,j. \quad (i \neq j \;\wedge\; Q[i,j] \;\wedge\; state[i] = won \rightarrow state[j] \neq won)$$

and *maximality*:

$$Max: \qquad \forall i \exists j \neq i. \quad (state[i] = lost \rightarrow Q[i,j] \;\wedge\; state[j] = won)$$

Note that the maximality property is not a $\forall$-property (rather, it's a $\forall\exists$-property) which is not directly covered by the Invisible Invariant methods. Note also that we are not dealing with the liveness property of the protocol, which claim that, with probability 1, every process eventually stops playing.

Another safety property we may wish to establish is that of stability of won/lost states, i.e., for every i,

$$Stbl: \quad \left(\begin{array}{l}(state[i] = won \rightarrow \quad (state[i] = won)) \wedge\\ (state[i] = lost \;\rightarrow \quad (state[i] = lost))\end{array}\right)$$

and that of "non-drift" among the phases of neighbors that may have been created by the "de-synchronization" of the protocol, i.e.,

$$Non\_drift: \quad \left(\begin{array}{l}\forall i,j. \quad (state[i] = playing \;\wedge\; state[j] = playing \;\wedge\; Q[i,j] \quad \rightarrow\\ \qquad\qquad (\mathbf{phase}[j] - \mathbf{phase}[i]) \bmod 3 \leq 1)\end{array}\right)$$

2.2 Leader Election Protocol

IEEE 1394 specifies a network allowing dynamic connection and disconnection of devices. At each point in time, the network is arranged as a tree, with devices as leaves. The *leader election* sub-protocol is invoked during a connection or disconnection event when, based on the new topology, a leader needs to be determined anew. Dynamic aspects of the network need not be modeled here since the leader election sub-protocol itself assumes a static network (i.e., following a connection/disconnection event).

As before, we model communication between nodes by shared variables. We let Q denote the adjacency matrix, and for each process i, we assign a boolean variable $done[i]$ denoting whether i still participates in the protocol or has determined its parent, a boolean $leader[i]$ which is set when i becomes the leader, and a boolean matrix $parent[1..N, 1..N]$ such that $parent[i, j]$ is set when j becomes the parent of i.

In our modeling of the protocol, we assume that each node i, in a single indivisible atomic step, can check all the $parent[1..N, i]$ variables and set $parent[i, j]$ and $leader[i]$ accordingly. This is different from the common synchronous modeling of the protocol that proceeds in send/receive phases, where at a send phase nodes can send "be my parent" requests and at receive phases nodes respond to such requests. There, *contention* may occur when two nodes send one another *be my parent* requests at the same phase. The atomicity assumption here bypasses root contention. As discussed later, the methods proposed here are applicable to less atomic versions that allow for contention.

$$
\begin{array}{l}
Q\colon \textbf{array}\ [1..N]\ \textbf{of array}\ [1..N]\ \textbf{of bool where}\ \forall i,j.Q[i,j] \leftrightarrow Q[j,i] \\
parent\colon \textbf{array}\ [1..N]\ \textbf{of array}\ [1..N]\ \textbf{of bool}\ \ \textbf{init}\ \forall i,j.\neg parent[i,j] \\
leader\colon \textbf{array}\ [1..N]\ \textbf{of bool init}\ \forall i.\neg leader[i] \\
done\colon :\ \textbf{array}\ [1..N]\ \textbf{of bool init}\ \forall i.\neg done[i] \\
\Big|\Big|_{i\neq j}\ P[i,j] :: \left[\begin{array}{l}
\textbf{while}\ \neg done[i]\ \textbf{do} \\
\quad \left[\begin{array}{l}
\textbf{if}\ \forall k \neq i.Q[i,k] \rightarrow parent[k,i]\ \textbf{then}\ (leader[i], done[i]) := (\mathbf{1},\mathbf{1}) \\
\textbf{elsif}\ (\neg parent[j,i] \wedge Q[i,j] \wedge \forall k \notin \{i,j\}.Q[i,k] \rightarrow parent[k,i]) \\
\qquad \textbf{then}\ (parent[i,j], done[i]) := (\mathbf{1},\mathbf{1})
\end{array}\right]
\end{array}\right]
\end{array}
$$

Fig. 3. Program LEADER-ELECT

The leader election protocol is shown in Fig. 3. For each node, the $parent$ matrix identifies which node is the parent of another node. There are $N(N-1)$ processes in the system, each corresponding to a pair $(i, j) \in [1..N]^2$ with $i \neq j$. Each such process, $P[i, j]$, repeatedly performs the following two steps while $done[i] \neq 1$:

1. The first if-statement executes if all nodes directly connected to i have i as their parent. In this case, i becomes the leader and sets $leader[i]$ to 1.
2. The second if-statement executes if (1) i and j are connected, (2) j has no parent, and (3) all other neighbors of i have i as parent. In this case, j becomes parent of i.

The protocol works as follows: Assume the underlying graph is a tree. Initially, all leaf nodes (and no internal node) can execute the second step. Then, the algorithm climbs

up the tree, each node executing the second step, until the root, which executes the first step, is reached.

If the original graph consists of a forest of trees, then a leader will be elected in each tree. If the original graph has non-tree connected components, then no leader will be elected in these components. The safety property of the protocol therefore states that each component contains at most one leader, formally stated by the following property:

$$\mathit{Unique}: \quad \forall i \neq j : \mathit{reachable}(i,j) \rightarrow \neg(\mathit{leader}[i] \wedge \mathit{leader}[j])$$

where for every $i, j \in [1..N]$, $\mathit{reachable}(i,j)$ holds if there is Q-path leading from i to j, i.e., if there are nodes $i_1, \ldots, i_k \in [1..N]$ such that $i_1 = i$, $i_k = j$, and for every $\ell = 1, \ldots, k-1$, $Q[i_\ell, i_{\ell+1}]$.

As discussed in the introduction, none of our old methods can be used to automatically verify this property. The method described in [16] fails since it depends on the $\mathit{reachable}$ predicate being based on a relation where each node has at most one successor, and Q, on which our current $\mathit{reachable}$ is based, does not satisfy this requirement.

3 Formal Model and Verifying Invariance

In this section we present our computational model, as well as the small model property that forms the basis of the verification method. Both model and property are derived from [5] and only differ in that the version here allows for matrix types (e.g., the Q and parent variables in Fig. 3).

3.1 Discrete Systems

As our computational model, we take a *discrete system* $S = \langle V, \Theta, \rho \rangle$, where

- V — A set of *system variables*. A *state* of S provides a type-consistent interpretation of the 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 all states over V.
- Θ — The *initial condition*: An assertion (state formula) characterizing the initial states.
- $\rho(V, V')$ — The *transition relation*: An assertion, relating the values V of the variables in state $s \in \Sigma$ to the values V' in an S-successor state $s' \in \Sigma$.

For an assertion ψ, we say that $s \in \Sigma$ is a ψ-state if $s \models \psi$.

A *computation* of a system S 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 an 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 $s_{\ell+1}[v]$.

3.2 Finite Network Systems

To allow the automatic decision of validity of assertions, we place further restrictions on the systems we study, leading to what is essentially the model of bounded discrete systems of [5] extended with an additional matrix type. For brevity, we describe here a simplified two-type model; the extension for the general multi-type case is straightforward. We allow the following data types parameterized by the positive integer N, intended to specify the size of the topology:

1. **bool**: boolean and finite-range scalars; With no loss of generality, we assume that all finite domain values are encoded as booleans.
2. **index**: $[1..N]$
3. Arrays of the types **index** $\mapsto$ **bool** (**bool** array) and **index** $\mapsto$ **index** $\mapsto$ **bool** (**bool** matrix)

Constants are introduced as variables with reserved names. Thus, we admit the boolean constants **0** and **1**, and **index** constants such as 1 and N. We often refer to an element of type **index** as a *node*. *Atomic formulas* are defined as follows:

- If x is a boolean variable, B is a **bool** array, and y is an **index** variable, then x and $B[y]$ are atomic formulas.
- If y_1 and y_2 are **index** variables and Q is a **bool** matrix, then $Q[y_1, y_2]$ is an atomic formula.
- If t_1 and t_2 are **index** terms, then $t_1 = t_2$ is an atomic formula.

A *restricted A-assertion* (resp. *restricted E-assertion*) is a formula of the form $\forall\vec{y}.\psi(\vec{x},\vec{y})$ (resp. $\exists\vec{y}.\psi(\vec{x},\vec{y})$) where $\vec{x}$ and $\vec{y}$ are lists of **index** variables, and $\psi(\vec{x},\vec{y})$ is a boolean combination of atomic formulae. A *restricted EA-assertion* is an assertion $\exists\vec{x}.\,\forall\vec{y}.\psi(\vec{x},\vec{y},\vec{u})$ where $\vec{u}$ is a list of **index** variables and $\forall\vec{y}.\psi(\vec{x},\vec{y},\vec{u})$ is a restricted A-assertion. *Restricted AE-assertion*s are similarly defined. As the initial condition Θ and the transition relation ρ we only allow restricted EA-assertions.

Let $\mathcal{V}$ be a *vocabulary* of typed variables, whose types are taken from the restricted type system allowed in a system. A *model* M for $\mathcal{V}$ consists of the following elements:

- A positive integer $N > 0$.
- For each boolean variable $b \in \mathcal{V}$, a boolean value $M[b] \in \{\mathbf{0}, \mathbf{1}\}$. It is required that $M[\mathbf{0}] = \mathbf{0}$ and $M[\mathbf{1}] = \mathbf{1}$.
- For each **index** variable $x \in \mathcal{V}$, a natural value $M[x] \in [1..N]$.
- For each boolean array $B \in \mathcal{V}$, a boolean function $M[B] : [1..N] \mapsto \{\mathbf{0}, \mathbf{1}\}$.
- For each boolean matrix $Q \in V$, a function $M[Q] : [1..N] \mapsto [1..N] \mapsto \{\mathbf{0}, \mathbf{1}\}$

We define the *size* of model M to be N.

The following theorem states that a restricted AE-assertion is valid iff it is valid over all models of a bounded size. It follows from a similar theorem of [5] (which does not deal with the boolean matrix data-type).

Theorem 1 (Small Model Property). *Let φ: $\forall\vec{y}\ \exists\vec{x}.\psi(\vec{y},\vec{x})$ be a closed restricted AE-assertion. Then φ is valid iff it is valid over all models of size not exceeding $|\vec{y}|$.*

3.3 Checking Invariance

Consider the INV proof rule of Fig. 1. When validating the premises of INV for restricted A-assertions p and φ, I3 is a boolean combination of A- and E-assertions, while I1 and I2 are AE-assertions. We now compute the cut-off bounds determined by the small model theorem to validate Premises I1 and I2. Assume that the assertions appearing in INV are of the form:

$$\begin{array}{ll} p\colon & (\forall u_1,\ldots,u_c.p_1(\vec{u})) \otimes \exists \vec{x}.p_2(\vec{x}) \\ \varphi\colon & (\forall u_1,\ldots,u_{n_\varphi}.\varphi_1(\vec{u})) \otimes \exists x_1,\ldots,x_{m_\varphi}.\varphi_2(\vec{x}) \\ \Theta\colon & \exists y_1,\ldots,y_a.\forall \vec{x}.t(\vec{y},\vec{x}) \\ \rho\colon & \exists y_1,\ldots,y_b.\forall \vec{x}.R(\vec{y},\vec{x}) \end{array}$$

where $\otimes \in \{\vee,\wedge\}$. I.e, p and φ are assertions that are disjunctions or conjunctions of a restricted A-assertion and a restricted E-assertion, and Θ and ρ are restricted EA-assertions. If p has free variables, then let $\hat{c}$ be c plus the number of free variables in p. Define $\hat{n}_\varphi$, $\hat{m}_\varphi$, $\hat{a}$, and $\hat{b}$ similarly. Theorem 1 now implies:

Corollary 1. *The premises of rule* INV *are valid over $S(N)$ for all $N > 1$ iff they are valid over $S(N)$ for all $N \leq \max\{\hat{a}+\hat{n}_\varphi, \hat{b}+\hat{n}_\varphi+\hat{m}_\varphi, \hat{m}_\varphi+\hat{c}\}$.*

3.4 Example: Verifying Program MIS

Consider Program MIS of Section 2. The system is of the form described in Section 3. Inspecting the structure of assertions Θ and ρ for this system, we see that $a = 0$ and $b = 2$ (since the transition relation is of the form $\exists i,j.\forall k.R(i,j,k)$).

For the property of independence we have $c = 2$. We instantiated the system to 4 processes and, using *proj-gen*, generated a candidate universal invariant $\varphi(i,j)$ with $n_\varphi = 2$ (and $m_\varphi = 0$). According to Corollary 1, it suffices to validate the premises of INV on models no larger than $\max\{2, 2+2, 2\} = 4$.

Next, let us consider the property of maximality which can be specified by the formula $p = \forall i \exists j.g(i,j)$, where $g(i,j)$ is given by

$$g(i,j): \qquad state[i] = lost \;\rightarrow\; (Q[i,j] \;\wedge\; state[j] = won)$$

Being an AE formula by itself, it is not implied by the invisibly derived inductive assertion. To establish this property, we directly apply rule INV with $\varphi = p = \forall i \exists j.g(i,j)$. On the face of it, this proof *seems* to fall outside the scope of the small model theorem since premise I2 has the form $(\forall i \exists j.g(i,j)) \wedge \rho \rightarrow (\forall u \exists v.g'(u,v))$, which is not of the required $\forall\exists$ form. We resolve this difficulty by observing that premise I2 is logically implied by the following $\forall\exists$ restricted assertion:

$$\forall u.((\exists j.g(u,j)) \wedge \rho \;\rightarrow\; \exists v.g'(u,v))$$

Hence, it is sufficient to check this stronger implication over the instance $S(4)$.

To show the stability of $state[i] = won$, we only need to show that it is preserved under transitions, i.e., that $\forall i.(state[i] = won \wedge \rho \rightarrow state'[i] = won)$. From the small model theorem (since ρ has two indices under existential quantification) it follows that

it suffices to check the above for $N_0 \leq 3$. The case of stability of $state[i] = lost$ is similar.

The property of "non-drift" is established in the standard way, since it is a universal assertion with $n_\varphi = 2$ which is implied by the invisibly derived invariant.

4 Reachability Avoidance

It is very often the case that safety properties of distributed systems include reachability predicates which are captured neither by Theorem 1 nor by the *proj-gen* heuristic. In this section we define the reachability properties we are interested in, and show a methodology that overcomes the challenges they pose to the Invisible Invariant method.

4.1 Safety Properties with Reachability

Let S be a distributed system with an underlying topology described by the adjacency matrix Q. Recall the $reachable(y_1, y_2)$ predicate denoting that y_2 is Q-reachable from y_1. In this section we study how to prove invariant properties of the type $(\alpha \otimes \beta)$, where α is a restricted A-assertion that allows for $reachable$ predicates, $\otimes \in \{\vee, \wedge\}$, and β is a restricted E-assertion (without reachability predicates).

For simplicity of exposition, we further restrict α to have a single occurrence of a $reachable$ predicate, both arguments of which are bound by the universal quantifier. Our results can be easily extended to cases where α has several occurrences of $reachable$, and to cases where some arguments of $reachable$ are free. An example of such a property is $Unique$ of program LEADER-ELECT in Section 2. There, β is trivial and α has a single $reachable$ predicate, both of whose arguments are under the scope of the universal quantification.

For the remaining part of this section we fix a safety property ϕ we wish to verify over S, where $\phi = \alpha \otimes \beta$ of the form above.

Let t be some **index** variable that does not appear free in either ϕ or the transition relation. Without loss of generality, assume that $\alpha\colon \forall i_1, \ldots, i_k.p(i_1, \ldots, i_k)$, where i_k is the first parameter of the (single) reachability predicate in α. Let $\alpha[t]$ be the formula $\forall i_1, \ldots, i_{k-1}.p(i_1, \ldots, i_{k-1}, t)$, and $\phi[t]$ be the formula $\alpha[t] \otimes \beta$. From the choice of t it follows that $S \models \phi[t]$ implies that $S \models \phi$.

For example, for property $Unique$ and $t = 1$, we obtain:

$$Unique[1]\colon \quad \forall j.j \neq 1 \wedge reachable(1, j) \rightarrow \neg(leader[1] \wedge leader[j])$$

4.2 Replacing Reachability with a First Order Predicate

The property $\phi[t]$ still contains a reachability predicate and its invariance cannot be handled by the method of Invisible Invariants. We next augment S with a "coloring protocol" and replace ϕ with a new property, ϕ^t, such that ϕ^t is of the form described in Section 3, such that when the augmented system satisfies ϕ^t we can conclude that $S \models \phi[t]$, and therefore $S \models \phi$.

The system and coloring protocol alternate once between "protocol" and "coloring" phases. While in the "protocol" phase, the system behaves like S, and the coloring

scheme is inactive. Similarly, while in the "coloring" phase, the system is inactive, and the coloring scheme behaves according to its protocol, C_t. An additional component, the "phase changer," determines which phase is first, and switches (once) between them. We shall return to the phase changer and first describe the coloring protocol.

The coloring protocol color_t, described in Fig. 4, propagates a marking starting at the node t. We assume a **boolean** array C_t that does not appear in S, all of whose entries are initially **0**, denoting that all nodes are uncolored. Once activated, the coloring protocol first sets $C_t[t]$, thus marking node t. Thereafter, when an uncolored node i has a colored neighbor j, $C_t[i]$ is set. The correctness of color_t is expressed in the following

color_t ::
  local C_t : **array** $[1..N]$ **of bool init** $\forall i.C_t[i] = \mathbf{0}$
  $\Big|\Big|_{i \neq j}$ $\big[$**if** $((i = t) \ \vee \ (Q[i,j] \ \wedge \ C_t[j] \ \wedge \ \neg C_t[i]))$ **then** $C_t[i] := 1\big]$

Fig. 4. System color_t

theorem, whose proof is by induction on the topology of the network:

Theorem 2. *Let $S[t] = S \| \mathsf{color}_t$. Then, for every node i, the following all hold:*

1. *$reachable(t,i)$ is S-valid iff it is $S[t]$-valid, i.e., both S and $S[t]$ have the same reachability relations;*
2. *$S[t] \models \quad (C_t[i] \rightarrow reachable(t,i))$, i.e., every colored node is reachable from t;*

Assume $\mathbf{phase}$ and $\mathbf{init\_phase}$ are variables not in S that can take on the values $\{\mathbf{color}, \mathbf{protocol}\}$. The *phase changer* PHASE is a module which composed with the S and color_t that is allowed to change the phase once, when a condition Ψ, which is an input to PHASE, is met. The module PHASE is described in Fig. 5. There, "$\mathbf{phase} := \neg\mathbf{phase}$" has the obvious meaning. In Subsection 4.3 we discuss how $\mathbf{init\_phase}$ and Ψ are initialized.

PHASE(Ψ) ::
  $\mathbf{phase}, \mathbf{init\_phase}$: $\{\mathbf{protocol}, \mathbf{color}\}$ **init** $\mathbf{phase} = \mathbf{init\_phase}$
  $\big[$**if** $(\Psi \ \wedge \ \mathbf{phase} = \mathbf{init\_phase})$ **then** $\mathbf{phase} := \neg\mathbf{phase}\big]$

Fig. 5. System PHASE(Ψ)

Let S' be the system S where each instruction is prefixed by "**if** $(\mathbf{phase} = \mathbf{protocol})$ **then** ...". Formally, if S is described by $\langle V, \Theta, \rho \rangle$ then S' is described by $\langle V \cup \{\mathbf{phase}\}, \Theta, \rho' \rangle$ where $\rho' = (\mathbf{phase} = \mathbf{protocol} \wedge \rho) \vee (\mathbf{phase} = \mathbf{color} \wedge \bigwedge_{v \in V} v = v')$. Similarly,

let $\mathsf{color}_t{}'$ be the system color_t where each instruction is prefixed by "**if** $(\mathbf{phase} = \mathbf{color})$ **then** ...". Then system S_{aug} is defined by the composition $S'\|\mathsf{color}_t{}'\|\text{PHASE}$.

The following claim follows immediately from the definition of S_{aug}:

Claim. Let ψ be a safety property over V. Then $S \models \psi$ iff $S_{aug} \models \psi$.

We next construct, from $\phi[t]$, a property ϕ^t such that $S^t \models \phi^t$ implies that $S_{aug} \models \phi[t]$ (which, according to the previous claim, implies that $S \models \phi[t]$). Recall that $\phi[t]$ is of the form $\forall\alpha[t] \otimes \exists\beta$ where the single reachability in $\phi[t]$ appears in α in the form $reachable(t, j)$. We first replace the $reachable(t, j)$ assertion in α by $C_t[j]$. If $reachable(t, j)$ appears in $\alpha[t]$ under positive polarity, we add to the resulting formula the disjunct

$$\exists j \neq k.Q[j, k] \ \wedge \ C_t[j] \ \wedge \ \neg C_t[k]$$

that captures the situation in which the coloring algorithm has not terminated yet. We take ϕ^t to be the resulting formula.

For example, under this transformation, $Unique[1]$ becomes:

$$Unique^1: \qquad \forall j.j \neq 1 \wedge C_1[j] \rightarrow \neg(leader[1] \wedge leader[j]) \tag{1}$$

The following theorem, whose proof is in Appendix A, establishes the soundness of the transformation.

Theorem 3.

$$S^t \models \phi^t \qquad \Longrightarrow \qquad S_{aug} \models \phi[t]$$

[Move theorem to tech report]

Note that ϕ^t is now of the form covered by Corollary 1. For example, to verify $Unique^1$, we have $a = 0$ (since the initial condition has no existential quantifiers), $b = 3$ since the transition relation of the augmented S^t has i and j under existential quantification, and t appears free in it, and $c = 2$, having j universally quantified and t free. Thus, for an auxiliary invariant φ, we would obtain a cutoff value of $\max\{n_\varphi, 3 + n_\varphi + m_\varphi, m_\varphi + 2\} = 3 + n_\varphi + m_\varphi$. We generated a φ with $n_\varphi = 2$ and $m_\varphi = 0$, and thus verified the premises of INV for every $N_0 \leq 5$.

4.3 Determining the Phase Alternation

There are two main choices to be made, namely, whether $\mathbf{init\_phase}$ is $\mathbf{protocol}$ or $\mathbf{color}$, and whether Ψ is trivially $\mathbf{1}$ or some non-trivial predicate. In our experiments, we used the trivial $\Psi = \mathbf{1}$ with $\mathbf{init\_phase}$ being both $\mathbf{protocol}$ or $\mathbf{color}$. As to non-trivial Ψ, we had to use it only once, in the verification of LEADER-ELECT, and then $\mathbf{init\_phase}$ was set to $\mathbf{protocol}$ and Ψ was defined as $leader[t]$. We recommend first trying to use a trivial $\Psi = \mathbf{1}$, and only if it fails under both choices of $\mathbf{init\_phase}$, to attempt some obvious Ψ's.

5 Evaluation

We have evaluated our method on a set of algorithms which, with the exception of Luby's maximal independent set algorithm, are based on versions found in [15]. The test cases consist of the leader election protocol used as the running example, a version of leader election that does not assume atomic parent request/acknowledge steps, as well as a distributed spanning tree algorithm. All experiments were evaluated using the TLV symbolic model-checker [17] on a Pentium 3 1GHz PC with 512Mb memory, and can be found at *http://www.cs.nyu.edu/acsys/dist-protocols/index.html*. A summary of runtime results is shown in Fig. 6. The rest of this section summarizes each test case.

Algorithm	**Runtime (seconds)**
Leader Election	5
Leader Election (alternate)	54
Spanning Tree	36
MIS	30

Fig. 6. Runtime Results

The alternate version of leader election allows for contention between nodes. While like the running example it treats the check over all of a node's neighbors as atomic, the assignment of parents is done in 2 phases, a *request* phase and an *acknowledgement* phase. Concretely, the matrix $parent$ is now of type
array $[1..N]$ **of array** $[1..N]$ **of** $\{\text{no}, \text{req}, \text{ack}\}$. Node j is considered the parent of i if $parent[i, j] = \text{ack}$.

For both versions of the leader election protocol, we verified the property $Unique$ defined in Section 2. For the alternate version we proved the additional property of *limited contention*, specifying that if neighboring nodes have requested parenthood from some neighbor, then the request is mutual:

$$\forall i \neq j, k, l : Q[i, j] \ \wedge \ parent[i, k] = \text{req} \ \wedge \ parent[j, l] = \text{req} \rightarrow k = l$$

Since this invariant effectively localizes contention in the protocol to two adjacent nodes, it serves as the basis for a liveness proof showing that any contention eventually converges with probability 1.

The spanning tree algorithm is similar to the coloring protocol color_t in that an arbitrary node is designated as the root, and nodes are added to the tree in a top-down, distributed fashion, starting at the root. For this algorithm we sought to verify the property that any node reachable from the root participates in the tree, unless tree propagation has not yet terminated, expressed as:

$$p\colon \ (\forall i, t : reachable(t, i) \rightarrow \text{in\_tree}[i]) \vee (\exists j \neq k : Q[j][k] \wedge \text{in\_tree}[j] \wedge \neg\text{in\_tree}[k])$$

where the boolean array in_tree denotes participation of nodes in the tree. However, we failed to generate an inductive auxiliary assertion that also implies this property. Instead, we did successfully verify that $\varphi \ \wedge \ p$ is an inductive invariant, where φ is the generated auxiliary assertion.

6 Conclusion and Discussion

We have described how the application of the method of Invisible Invariants to distributed protocols with an arbitrary fixed topology.

Contrary to common belief, we found that the extension of the method to arbitrary, as opposed to trivial, topologies is rather straightforward (as demonstrated by the verification of Luby's MIS protocol). Yet, the correctness of many such protocols is specified by means of reachability predicates, which cannot be captured by the invisible invariant method. We present a simple coloring augmentation that allows, in many cases, to replace reachability predicates by simpler first order predicates that can be dealt with by the invisible invariant methods.

There are several weaknesses to our scheme:

- Many distributed systems are modeled as synchronous, i.e., their transition relation is an AEA-assertion. This is beyond the power of our small model theorem, hence we "de-synchronize" them. We would like to identify the types of synchronous systems our method applies to;
- Our scheme depends on running the "system" and the "coloring," one after the other, switching once from one to the other at some point. Often, this point is nondeterministic and the only choice is which protocol to run first. Yet, it is sometimes the case that the switch can happen only when some condition is attained. Here the method is not fully automatic since the user has to guess the condition, which requires some familiarity with the protocol.
- Our scheme is dependent on the invisible invariant method, and is restricted by its power. Being a BDD-based method, the size of the instantiation of the system required may be too large to handle. In addition, *proj-gen* can only generate invariants of certain syntactic type, and it may be the case that the invariants needed are beyond its power. (For example, *proj-gen* generates restricted EA-invariants, is is extremely limited in the AE-invariants it generates.)

Yet, in spite of the restrictions, we succeeded to automatically verify, for the first time, some classical examples that have been thoroughly studied in the literature.

We are hopeful that our coloring augmentation can be used in verification of other systems too, for example, pointer systems. We are currently working on extending the system to handle mobile networks.

Acknowledgement: We would like to thank Shuvendu Lahiri, who brought the Leader Election protocol to our attention, and Yi Fang who pointed out that our existing small model theorem can be applied to adjacency matrices.

References

1. Shankar, N., Owre, S., Rushby, J.M.: A tutorial on specification and verification using PVS. Technical report (1993)
2. Bjørner, N., Browne, I., Chang, E., Colón, M., Kapur, A., Manna, Z., Sipma, H., Uribe, T.: STeP: The Stanford Temporal Prover, User's Manual. Technical Report STAN-CS-TR-95-1562, Computer Science Department, Stanford University (1995)

3. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Safety. Springer Verlag, New York (1995)
4. Pnueli, A., Ruah, S., Zuck, L.: Automatic deductive verification with invisible invariants. In: TACAS'01, LNCS 2031 (2001) 82–97
5. Arons, T., Pnueli, A., Ruah, S., Xu, J., Zuck, L.: Parameterized verification with automatically computed inductive assertions. In: CAV'01, LNCS 2102 (2001) 221–234
6. Balaban, I., Fang, Y., Pnueli, A., Zuck, L.: IIV: An invisible invariant verifier. In: Computer Aided Verification (CAV). (2005)
7. Luby, M.: A simple parallel algorithm for the maximal independent set problem. SIAM Journal of Computing **15**(4) (1986) 1036–1053
8. Zuck, L., Pnueli, A.: Model checking and abstraction to the aid of parameterized systems. Computer Languages, Systems, and Structures **30**(3–4) (2004) 139–169
9. Baukus, K., Lakhnech, Y., Stahl, K.: Parameterized verification of a cache coherence protocol safety and liveness. In: Proceedings of the 6th International Conference on Verification, Model Checking, and Abstract Interpretation. (2002) 317–330
10. Lahiri, S., Bryant, R.: Constructing quantified invariants via predicate abstraction. In: Proceedings of the 5th International Conference on Verification, Model Checking, and Abstract Interpretation. (2004) 267–281
11. Romijn, J.M.T.: A timed verification of the IEEE 1394 leader election protocol. In Gnesi, S., Latella, D., eds.: Proceedings of the Fourth International ERCIM Workshop on Formal Methods for Industrial Critical Systems (FMICS'99). (1999) pages 3–29
12. Devillers, M., Griffioen, W., Romijn, J., Vaandrager, F.: Verification of a leader election protocol: Formal methods applied to IEEE 1394. Technical Report CSI-R9728, Computing Science Institute, Nijmegen (1997)
13. Daws, C., Kwiatkowska, M., Norman, G.: Automatic verification of the IEEE 1394 root contention protocol with KRONOS and PRISM. In Cleaveland, R., Garavel, H., eds.: Proc. 7th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'02). Volume 66.2 of Electronic Notes in Theoretical Computer Science., Elsevier (2002)
14. Lev-Ami, T., Immerman, N., Reps, T.W., Sagiv, S., Srivastava, S., Yorsh, G.: Simulating reachability using first-order logic with applications to verification of linked data structures. In: CADE. (2005) 99–115
15. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1996)
16. Balaban, I., Pnueli, A., Zuck, L.: Shape analysis by predicate abstraction. In: Proceedings of the 6th International Conference on Verification, Model Checking, and Abstract Interpretation. (2005) 164–180
17. Shahar, E.: The TLV Manual. (2000) http://www.cs.nyu.edu/acsys/tlv.

A Proof of Theorem 3

Assume that ϕ^t is S^t-valid. From Theorem 2 it suffices to show that every S'-state satisfies $\phi[t]$. Assume that the boolean connective, $\otimes$, in $\phi[t]$ is a disjunction (the case of a conjunction is similarly established). Let s be an S^t-state so that s satisfies ϕ^t. If the reachability assertion in α is of negative polarity, then from part (2) of Theorem 2 it follows that $s \models \neg reachable(t,i) \rightarrow \neg C_t[i]$. Hence, if ϕ^t holds in s, then so does $\phi[t]$.

Assume therefore that the reachability assertion in α, $reachable(t,i)$, is in positive polarity. If in s, $C_t[i]$ is set, or if $\neg reachable(t,i)$ holds, then obviously s satisfied ϕ. Assume therefore that $s \models reachable(t,i) \wedge \neg C_t[i]$. It can be easily shown from

System color_t that there exist nodes j and k such that $s \models Q[i,j] \wedge C_t[j] \wedge \neg C_t[k]$. Let $\sigma\colon s_0, s_1, \ldots$ be an S^t-computation, starting with s that include no idle steps. Moreover, if $s \models \neg\mathbf{color}$, then let the first transition in σ be one that sets $\mathbf{color}$. Note that such a σ must exist. Since once $\mathbf{color}$ holds the evaluation of $\phi[t]$ remains invariant, in that if for some j, $s_j \models \phi[t]$, we can conclude that $s = s_0 \models \phi[t]$.

The computation σ must have a state s_k in which the coloring terminates. Since ϕ^t is S^t-valid, ϕ^t holds in s_k. Also, since in s_k, $\mathit{reachable}(t,i) \leftrightarrow C_t[i]$, $\phi[t]$ holds in s_k. It now follows that $\phi[t]$ holds in s. $\square$