Liveness with Invisible Ranking*

Yi Fang[1], Nir Piterman[2], Amir Pnueli[21], and Lenore Zuck[1]

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

Abstract. The method of Invisible Invariants was developed originally in order to verify safety properties of parameterized systems fully automatically. Roughly speaking, the method is based on a *small model property* that implies it is sufficient to prove some properties on small instantiations of the system, and on a heuristic that generates candidate invariants. Liveness properties usually require well founded ranking, and do not fall within the scope of the small model theorem. In this paper we develop novel proof rules for liveness properties, all of whose proof obligations are of the correct form to be handled by the small model theorem. We then develop abstractions and generalization techniques that allow for fully automatic verification of liveness properties of parameterized systems. We demonstrate the application of the method on several examples.

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. Establish the logical validity of the premises, using the auxiliary constructs identified in step 1.

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 [21] or STeP [4]. The difficulty in the execution of these two steps is the main reason why deductive verification is not used more extensively.

A representative case is the verification of invariance properties using the *invariance rule* of [17]. In order to prove that assertion r is an invariant of program P, the rule requires coming up with an auxiliary assertion φ which is *inductive* (i.e. is implied by the

* This research was supported in part by the Minerva Center for Verification of Reactive Systems, the European Community IST project "Advance", the Israel Science Foundation (grant no. 106/02-1), and NSF grant CCR-0205571.

initial condition and is preserved under every computation step) and which strengthens (implies) r.

In [19, 2] 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. In this paper we generalize the method of invisible invariants to deal with liveness properties.

The method of invisible invariants is based on two main ideas:

It is often the case that the auxiliary assertion for a parameterized system has the form $\varphi : \forall i.q(i)$ (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 system $S(N_0)$, we compute the set of reachable states *reach* using a symbolic model checker. Let r_1 be the projection of *reach* 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 *projection and generalization*.

Having obtained a candidate for the inductive assertion φ, we still have to check the validity of the three premises of the invariance rule. 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), and the fact that our candidate inductive assertions have the form $\forall \vec{i}.q(\vec{i})$, we managed to prove a *small model* theorem. According to this theorem, there exists a (small) bound N_0 such that the premises of the invariance rule are valid for every N iff they are valid for all $N \leq N_0$. This enables us to use BDD techniques in order to check the validity of the premises. This theorem is based on the fact that, under the above assumptions, all three premises 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 which may refer only to the global variables and the local variables of $P[i]$ and $P[j]$.

Being able to validate the premises on $S[N_0]$ has the additional important advantage that the user does not have to see the automatically generated auxiliary assertion φ. This assertion is generated as part of the procedure and is immediately used in order to validate the three premises of the rule. This is advantageous because, being generated by symbolic BDD techniques, its representation 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 the auxiliary invariant, we refer to this method as the method of *invisible invariants*.

In this paper, we extend the method of invisible invariants to apply to proofs of the second most important class of properties which the class of *response properties*. These are liveness properties which can be specified by the temporal formula $\Box(q \rightarrow \Diamond r)$ (also written as $q \Rightarrow \Diamond r$) and guarantees that any q-state is eventually followed by a r-state. To do so, we consider a certain variant of rule WELL [16] which establishes the validity of response properties under the assumption of *justice* (weak fairness). As is well known to users of this and similar rules, such a proof requires the generation of two kinds of auxiliary constructs: "helpful" assertions which characterize the states at which a certain transition is helpful in promoting progress towards the goal (r), and *ranking functions* which measure the progress towards the goal.

In order to make the *project-and-generalize* technique applicable to the automatic generation of the ranking functions, we developed a special variant of rule WELL, to which we refer as rule DISTRANK. In this version of the rule, we associate with each potentially helpful transition τ_i an individual ranking function $\delta_i : \Sigma \mapsto [0..c]$, mapping states to integers in a small range $[0..c]$, where c is a fixed small constant, independent of the parameter N. The global ranking function can be obtained by forming the multi-set $\{\delta_i\}$. In most of the examples we considered, it was sufficient to take the range of the individual functions to be $\{0, 1\}$, which enable us to view each δ_i as an assertion, and generate it automatically using the *project-and-generalize* techniques.

The paper is organized as follows: In Section 2, we present the general computational model of FDS and the restrictions which enable the application of the invisible auxiliary constructs methods. In this section we also review the small model property which enables us to automatically validate the premises of the various proof rules.

In Section 3 we introduce the new DISTRANK proof rule, explain how we automatically generate ranking and helpful assertions for the parameterized case, and demonstrate the techniques on the two examples of a TOKEN-RING and the BAKERY algorithms. The method introduced in this section is adequate for all the cases in which the set of reachable states can be satisfactorily over-approximated by an assertion of the form $\forall i.u(i)$ and both the helpful assertion h_i and the individual ranking function δ_i representable by an assertion of the form $\alpha(i)$ which only refers to the local variables of process $P[i]$.

Not all examples can be handled by assertions which depend on a single parameter. In Section 4, we consider cases whose verification requires a characterization of the reachable states by an assertion of the form $\forall i.u(i) \ \wedge \ \exists j.e(j)$. In a future paper we will consider helpful assertions h_i which have the form $h_i : \forall j \neq i.\psi(i, j)$. With these extensions we can handle another version of the BAKERY algorithm. A variant of this method has been successfully applied to Szymanski's N-process mutual exclusion algorithm.

Related Work. In [20] we introduced the method of "counter-abstraction" to automatically prove liveness properties of parameterized systems. Counter-abstraction is an instance of data-abstraction [13] and has proven successful in instances of systems with a trivial (star or clique) topologies and a small state-space for each process. The work there is similar to counter abstraction is the work of the PAX group (see, e.g., [3]) which is based on the method of *predicate abstraction* [5]. While there are several differences between the two approaches, both are not "fully automatic" in the sense that the user has to provide the system with abstraction methodology.

In [22, 14] we used the method of *network invariants* [13] to prove liveness properties of parameterized systems. While extremely powerful, the main weakness of the method is that the abstraction is performed manually by the user.

The problem of uniform verification of parameterized systems is, in general, undecidable [1]. One approach to remedy this situation, pursued, e.g., in [8], is to look for restricted families of parameterized systems for which the problem becomes decidable.

Many of these approaches fail when applied to asynchronous systems where processes communicate by shared variables.

Another approach is to look for sound but incomplete methods. Representative works of this approach include methods based on: explicit induction ([9]), network invariants that can be viewed as implicit induction ([15]), abstraction and approximation of network invariants ([6]), and other methods based on abstraction ([10]). Other methods include those relying on "regular model-checking" (e.g., [12]) that require special *acceleration* procedures and thus involve user ingenuity and intervention, methods based on symmetry reduction (e.g., [11]), or compositional methods (e.g., ([18]) that combine automatic abstraction with finite-instantiation due to symmetry. These works, from which we have mentioned only few representatives, require the user to provide auxiliary constructs and thus do not provide for fully automatic verification of parameterized systems.

Less related to our work is the work in [7] which presents methods for obtaining ranking functions for sequential programs.

2 The Model

As our basic computational model, we take a *fair discrete system* (FDS) $S = \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$, where

- V — A set of *system variables*. 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 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 $s' \in \Sigma$.
- $\mathcal{J}$ — A set of *justice* (*weak fairness*) requirements: Each justice requirement is an assertion; A computation must include infinitely many states satisfying the requirement.
- $\mathcal{C}$ — A set of *compassion (strong fairness)* requirements: Each compassion requirement is a pair $\langle p, q \rangle$ of state assertions; A computation should include either only finitely many p-states, or infinitely many q-states.

A *computation* of an FDS 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 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 $s_{\ell+1}[v]$.
- *Justice* — for every $J \in \mathcal{J}$, σ contains infinitely many occurrences of J-states.
- *Compassion* – for every $\langle p, q \rangle \in \mathcal{C}$, either σ contains only finitely many occurrences of p-states, or σ contains infinitely many occurrences of q-states.

For simplicity, all of our examples will contain no compassion requirements, i.e., $\mathcal{C} = \emptyset$. Most of the methods can be generalized to deal with systems with a non-empty set of compassion requirements.

2.1 Bounded Fair Discrete Systems

To allow the application of the invisible constructs methods, we place further restrictions on the systems we study, leading to the model of *fair bounded discrete systems* (FBDS), that is essentially the model of bounded discrete systems of [2] augmented with fairness. For brevity, we describe here a simplified two-type model; the extension for the general multi-type case is straightforward.

Let $N \in \mathbb{N}^+$ be the *system's parameter*. We allow the following data types:

1. $\mathbf{type}_0$: the set of boolean and finite-range scalars (also denoted **bool**);
2. $\mathbf{type}_1$: a scalar data type that includes integers in the range $[1..N]$;
3. $\mathbf{type}_2$: a scalar data type that includes integers in the range $[0..N]$; and
4. arrays of the type $\mathbf{type}_1 \mapsto \mathbf{type}_i$ for $i = 0, 2$.

For simplicity, we allow the system variables to include any number of variables of types $\mathbf{type}_0$, $\mathbf{type}_1$, and $\mathbf{type}_1 \mapsto \mathbf{type}_0$, but at most a single variable of type $\mathbf{type}_1 \mapsto \mathbf{type}_2$, and no variables of type $\mathbf{type}_2$.

Atomic formulas may compare two variables of the same type. E.g., if y and y' are $\mathbf{type}_1$ variables, and z is a $\mathbf{type}_1 \mapsto \mathbf{type}_2$, then $y = y'$ and $z[y] < z[y']$ are both atomic formulas. For $z : \mathbf{type}_1 \mapsto \mathbf{type}_2$ and $y : \mathbf{type}_1$, we also allow the special atomic formula $z[y] > 0$. We refer to quantifier-free formulas obtained by boolean combinations of such atomic formulas as *restricted assertions*.

As the initial condition Θ, we allow assertions of the form $\forall i.u(i) \wedge \exists j.e(j)$, where $u(i)$ and $e(j)$ are restricted assertions.

As the transition relation ρ, we allow assertions of the form $\exists \vec{i} \forall \vec{j}.\psi(\vec{i}, \vec{j})$ for a restricted assertion $\psi(\vec{i}, \vec{j})$.

The allowed justice requirements are restricted assertions which may be parameterized by a $\mathbf{type}_1$ index.

All quantified variables must be of type $\mathbf{type}_1$.

Example 1 (The Token Ring Algorithm).
Consider program TOKEN-RING in Fig. 1, which is a mutual exclusion algorithm for any N processes.

In this version of the algorithm, the global variable *tloc* represents the current location of the token. Location ℓ_0 constitutes the non-critical section which may non-deterministically exit to the trying section at location ℓ_1. While being in the non-critical section, a process guarantees to move the token to its right neighbor, whenever it receives it. This is done by incrementing *tloc* by 1, modulo N. At the trying section, a process $P[i]$ waits until it received the token which is signaled by the condition $tloc = i$.

$$
\begin{array}{l}
\quad\quad\quad\quad \textbf{in } N \ : \textbf{natural where } N > 1 \\
\quad\quad\quad\quad\quad \textit{tloc} : [1..N] \\
\overset{N}{\underset{i=1}{\Big\|}} P[i] :: \left[\begin{array}{l} \textbf{loop forever do} \\ \left[\begin{array}{l} \ell_0 : \textbf{if } \textit{tloc} = i \textbf{ then } \textit{tloc} := \textit{tloc} \oplus_N \mathbf{+1} \\ \quad\ \ \textbf{go to } \{\ell_0, \ell_1\} \\ \ell_1 : \textbf{await } \textit{tloc} = i \\ \ell_2 : \textbf{Critical} \end{array} \right] \end{array} \right]
\end{array}
$$

Fig. 1. Program TOKEN-RING

Following is the FBDS corresponding to program TOKEN-RING:

$$
\begin{array}{l}
V : \begin{cases} \textit{tloc} : [1..N] \\ \pi : \quad \textbf{array}[1..N] \textbf{ of } [0..2] \end{cases} \\
\Theta : \forall i.\pi[i] = 0 \\
\rho : \ \exists i : \forall j \neq i : (\pi'[j] = \pi[j]) \wedge \\
\quad\quad \left[\begin{array}{ll} & \pi[i] = 0 \ \wedge \ \textit{tloc} = i \ \wedge \ \textit{tloc}' = i \oplus_N +1 \ \wedge \ \pi'[i] \in \{0, 1\} \\ \vee & \textit{tloc}' = \textit{tloc} \ \wedge \ \left(\begin{array}{l} \quad \pi'[i] = \pi[i] \\ \vee \ \pi[i] = 0 \ \wedge \ \textit{tloc} \neq i \ \wedge \ \pi'[i] = 1 \\ \vee \ \pi[i] = 1 \ \wedge \ \textit{tloc} = i \ \wedge \ \pi'[i] = 2 \\ \vee \ \pi[i] = 2 \ \wedge \ \pi'[i] = 3 \end{array} \right) \end{array} \right] \\
\mathcal{J} : \left(\begin{array}{lll} \{J_0[i] : \neg(\pi[i] = 0 \ \wedge \ \textit{tloc} = i) & | \ i \in [1..N]\} & \cup \\ \{J_1[i] : \neg(\pi[i] = 1 \ \wedge \ \textit{tloc} = i) & | \ i \in [1..N]\} & \cup \\ \{J_2[i] : \quad \pi[i] \neq 2 & | \ i \in [1..N]\} & \end{array} \right)
\end{array}
$$

Note that *tloc* is a variable of type $\textbf{type}_1$, while the program counter π is a variable of type $\textbf{type}_1 \mapsto \textbf{type}_0$.

Strictly speaking, the transition relation as presented above does not fully conform to the definition of a restricted assertion since it contains the atomic formula $\textit{tloc}' = i \oplus_N +1$. However, this can be rectified by a two-stage reduction. First, we replace $\textit{tloc}' = i \oplus_N +1$ by $(i < N \ \wedge \ \textit{tloc}' = i + 1) \ \vee \ (i = N \ \wedge \ \textit{tloc}' = 1)$. Then, we replace the formula $\exists i : \forall j \neq i : (\ldots \textit{tloc}' = i + 1 \ldots)$ by
$\exists i, i_1 : \forall j \neq i, j_1 : (j_1 \leq i \ \vee \ i_1 \leq j_1) \ \wedge \ (\ldots \textit{tloc}' = i_1 \ldots)$ which guarantees that $i_1 = i + 1$.

Let α be an assertion over V, and R be an assertion over $V \cup V'$, which can be viewed as a transition relation. We denote by $\alpha \circ R$ the assertion characterizing all state which are R-successors of α-states. We denote by $\alpha \circ R^*$ the states reachable by an R-path of length zero or more from an α-state.

2.2 The Small Model Theorem

Let $\varphi : \forall \vec{i} \exists \vec{j}.R(\vec{i}, \vec{j})$ be an AE-formula, where $R(\vec{i}, \vec{j})$ is a restricted assertion which refers to the state variables of a parameterized FBDS $S(N)$ in addition to the quantified variables $\vec{i}$ and $\vec{j}$. Let N_0 be the number of universally quantified and free $\textbf{type}_1$ variables appearing in R. The following claim (stated first in [19] and extended in [2]) provides the basis for the automatic validation of the premises in the proof rules:

Claim 1 (Small model property).
Formula φ is valid iff it is valid over all instances $S(N)$ for $N \leq N_0$.

Proof Sketch
To prove the claim, we consider the negation of the formula, given by $\psi : \exists\vec{i}\forall\vec{j}.\neg R(\vec{i},\vec{j})$, and show that if ψ is satisfiable, then it is satisfiable in a state of a system $S(N)$ for some $N \leq N_0$. Assume that ψ is satisfiable in some state of an instance $S(N_1)$. Let $u_1 < u_2 < \cdots < u_k$ be the sequence of pairwise distinct values of **type**$_1$ variables which appear free or existentially quantified within ψ, sorted in ascending order. Obviously $k \leq N_0$. We can construct a state in the system $S(k)$ by mapping all **type**$_1$-values equal to u_i into i. It can be shown that this state satisfies ψ. □

In case the formula φ refers to **type**$_1$-constants such as 1 or N, they should be added to the count of variables which yields N_0. Similarly, if it refers to an expression such as $i+1$, we should add 1 more variable which participates in the re-expression of $i+1$.

3 The Method of Invisible Ranking

In this section we present a new proof rule for liveness properties of an FBDS that allows, in some cases, to obtain an automatic verification of liveness properties for systems of any size. We first describe the new proof rule. We then present methods for the automatic generation of the auxiliary constructs required by the rule. We illustrate the application of these method on the example of TOKEN-RING as we go along.

3.1 A Distributed Ranking Proof Rule

In Fig. 2, We present proof rule DISTRANK (for *distributed ranking*), for verifying response properties. This rule assumes that the system to be verified has only justice requirements (in particular, it has no compassion requirements.)
The rule is configured to deal directly with parameterized systems. Typically, the parameter domain provides a unique identification for each transition, and will have the form $\mathcal{P}(N) = [0..k] \times N$ for some fixed k. For example, in program TOKEN-RING, $\mathcal{P}(N) = [0..2] \times N$, where each justice transition can be identified as $J_m[i]$ for $m \in [0..2]$, and $i \in [1..N]$. In the rule, assertion φ is an invariant assertion characterizing all the reachable states. Assertion $pend$ characterizes the states which can be reached from a reachable q-state by a r-free path. For each justice requirement J_p, assertion h_p characterizes the states at which this justice requirement is *helpful*. That is, these are the states whose every J_p-satisfying successor leads to a progress towards the goal, which is expressed by immediately reaching the goal or a decrease in on of the ranking functions, as stated in premise D5. The ranking functions δ_p are used in order to measure progress towards th goal.

Premise D1 guarantees that any reachable q-state satisfies r or $pend$. Premise D2 guarantees that any successor of a $pend$-state also satisfies r or $pend$. Premise D3 guarantees that any $pend$-state has at least one justice requirement for which it is helpful.

For a parameterized system with a parameter domain $\mathcal{P} = \mathcal{P}(N)$
set of states $\Sigma(N)$,
justice requirements $\{J_p \mid p \in \mathcal{P}\}$,
invariant assertion φ,
assertions $q, r, pend$ and $\{h_p \mid p \in \mathcal{P}\}$,
and ranking functions $\{\delta_p : \Sigma \to \{0,1\} \mid p \in \mathcal{P}\}$

$$
\begin{array}{lll}
\text{D1. } q \wedge \varphi & \to & r \vee pend \\
\text{D2. } pend \wedge \rho & \to & r' \vee pend' \\
\text{D3. } pend & \to & \bigvee_{p \in \mathcal{P}} h_p \\
\text{D4. } pend \wedge \rho & \to & r' \vee \bigwedge_{p \in \mathcal{P}} \delta_p \geq \delta'_p \\
\text{For every } p \in \mathcal{P} & & \\
\text{D5. } h_p \wedge \rho & \to & r' \vee h'_p \vee \delta_p > \delta'_p \\
\text{D6. } h_p & \to & \neg J_p \\
\hline
 & q \Rightarrow \Diamond r &
\end{array}
$$

Fig. 2. The liveness rule DISTRANK

Premise D4 guarantees that a step between two $pend$-states cannot cause any of the ranking functions to increase. Premise D5 guarantees that taking a step from an h_p-state leads into a state which either already satisfies the goal r, or causes the rank δ_p to decrease, or is again an h_p-state. Premise D6 guarantees that all h_p-states do not satisfy J_p. Together, premises D5 and D6 imply that the computation cannot stay in h_p forever, because this will violate the requirement of justice w.r.t J_p. Therefore, the computation must eventually move to a $\neg h_p$-state which will cause δ_p to decrease. Since we have only finitely many δ_p and until the goal is reaches they never increase, we can conclude that eventually we must reach r.

3.2 Automatic Generation of the Auxiliary Constructs

We now proceed to show how all the auxiliary constructs necessary for the application of rule DISTRANK can be automatically generated. Note that we have to construct a symbolic version of these constructs, so that the rule can be applied to a generic N.

For simplicity, we assume that the response property we wish to establish only refers to the local variables of process $P[1]$. A case in point is the verification of the property $at\_\ell_1[1] \Rightarrow \Diamond\, at\_\ell_2[1]$ for program TOKEN-RING. This property claims that every state in which process $P[1]$ is at location ℓ_1 is eventually followed by a state in which process $P[1]$ is at location ℓ_2. There are generalizations of the approach to the case that we wish to prove a response property which refers to the local variables of process $P[z]$ for an arbitrary symbolic $z \in [1..N]$. Assume also that the parameter domain has the form $[0..m] \times N$.

In this case (that the special process is $P[1]$), we would expect the constructs to have the symbolic forms $\varphi : \forall i.\varphi^{\mathrm{A}}(i)$ and $pend : pend^{\mathrm{A}}(1) \wedge \forall i.pend^{\mathrm{A}}(i)$. For each $k \in [0..m]$, we need to compute $h^{\mathrm{A}}_k[1], \delta^{\mathrm{A}}_k[1]$, and the generic $h^{\mathrm{A}}_k[i], \delta^{\mathrm{A}}_k[i]$, which should be symbolic in i and apply for all i, $1 < i \leq N$. All generic constructs are allowed to refer to the global variables and to the variables local to $P[1]$. We will consider each of the auxiliary constructs and provide a recipe for its generation. The recipe will be illustrated on the case of program TOKEN-RING. The construction uses the instantiation

$S(N_0)$ for an appropriately chosen N_0, selected as explained below. We denote by $\Theta_{_C}$ and $\rho_{_C}$ the initial condition and transition relation for $S(N_0)$. The construction begins by computing the *concrete* auxiliary constructs for $S(N_0)$ which we denote by $\varphi_{_C}$, $pend_{_C}$, $h_k^{\mathrm{C}}[j]$, and $\delta_k^{\mathrm{C}}[j]$. We then use projection and generalization in order to derive the symbolic (*abstract*) versions of these constructs: $\varphi_{_A}$, $pend_{_A}$, $h_k^{\mathrm{A}}[j]$, and $\delta_k^{\mathrm{A}}[j]$.

Computing $\boldsymbol{\varphi_{_A}}$: Compute the assertion $reach_{_C} = \Theta_{_C} \circ \rho_{_C}^*$ characterizing all states reachable by system $S(N_0)$. We take $\varphi^{\mathrm{A}}(i) = reach_{_C}[3 \mapsto i]$, which is obtained by projecting $reach_{_C}$ on index 3, and then generalizing 3 to i.

For example, in TOKEN-RING(6), $reach_{_C} = \bigwedge_{j=1}^{6}(at\_\ell_{0,1}[j] \vee tloc = j)$. The projection of $reach_{_C}$ on $j = 3$ yields $(at\_\ell_{0,1}[3] \vee tloc = 3)$. The generalization of 3 to i yields $\varphi^{\mathrm{A}}(i) : at\_\ell_{0,1}[i] \vee tloc = i$. Note that when we generalize, we should generalize not only the values of the variables local to $P[3]$ but also the case that the global variable, such as *tloc*, has the value 3. The choice of 3 as the generic value is arbitrary. Any other value would do as well, but we prefer indices different from $1, N$.

Computing $\boldsymbol{pend_{_A}}$: Compute the assertion $pend_{_C} = (reach_{_C} \wedge q \wedge \neg r) \circ (\rho_{_C} \wedge \neg r)^*$, characterizing all states which can be reached from a reachable $q \wedge \neg r$-state by a r-free path. Then we take $pend^{\mathrm{A}}(1) = pend_{_C}[1 \mapsto 1]$, and $pend^{\mathrm{A}}(i) = pend_{_C}[1 \mapsto 1, 3 \mapsto i]$.

Thus, for TOKEN-RING(6), $pend_{_C} = reach_{_C} \wedge at\_\ell_1[1]$. We therefore take $pend^{\mathrm{A}}(1) : at\_\ell_1[1]$ and $pend^{\mathrm{A}}(i) : at\_\ell_1[1] \wedge (at\_\ell_{0,1}[i] \vee tloc = i)$.

Computing $\boldsymbol{h_k^{\mathrm{A}}[i]}$: First, we compute the concrete helpful assertions $h_k^{\mathrm{C}}[j]$. This is based on the following analysis. Assume that *set* is an assertion characterizing a set of states, and let J be some justice requirement. We wish to identify the subset of states ϕ within *set* for which the transition associated with J is an *escape* transition. That is, any application of this transition to a ϕ-state takes us out of *set*. Consider the fix-point equation:

$$\phi \quad = \quad set \ \wedge \ \neg J \ \wedge \ AX(\phi \ \vee \ \neg set) \tag{1}$$

The equation states that every ϕ-state must satisfy $set \ \wedge \ \neg J$, and that every successor of a ϕ-state is either a ϕ-state or lies outside of *set*. In particular, all J-satisfying successors of a ϕ-state do not belong to *set*. By taking the maximal solution of this fix-point equation, denoted $\nu\phi(set \wedge \neg J \ \wedge \ AX(\phi \vee \neg set))$, we compute the subset of states which are helpful for J within *set*.

Following is an algorithm which computes the concrete helpful assertions $\{h_k^{\mathrm{C}}[j]\}$ corresponding to the justice requirements $\{J_k^{\mathrm{C}}[j]\}$ of system $S(N_0)$. For simplicity, we will use $p \in \mathcal{P}(N_0)$ as a single parameter.

> **for each** $p \in \mathcal{P}(N_0)$ **do** $h_p := 0$
> $set := pend_{_C}$
> **for all** $p \in \mathcal{P}(N_0)$ **satisfying** $\nu\phi(set \wedge \neg J_p \ \wedge \ AX(\phi \vee \neg set)) \neq 0$ **do**
> $\left[\begin{array}{l} h_p := h_p \ \vee \ \nu\phi(set \wedge \neg J_p \ \wedge \ AX(\phi \vee \neg set)) \\ set := set \ \wedge \ \neg h_p \end{array}\right]$

The "**for all** $p \in \mathcal{P}(N_0)$" iteration terminates when it is no longer possible to find a $p \in \mathcal{P}(N_0)$ which satisfies the non-emptiness requirement. The iteration may choose the same p more than once. Ideally, $set = 0$ when the iteration terminates, i.e., for each of the states covered under $pend_{_C}$ there exists a helpful justice requirement which causes it to progress.

Having found the concrete $h^{\mathrm{C}}_k[j]$, we proceed to project and generalize as follows: for each $k \in [0..m]$, we let $h^{\mathrm{A}}_k[1] = h^{\mathrm{C}}_k[1][1 \mapsto 1]$ and $h^{\mathrm{A}}_k[i] = h^{\mathrm{C}}_k[3][1 \mapsto 1, 3 \mapsto i]$.

Applying this procedure to TOKEN-RING(6), we obtain the following symbolic helpful assertions:

	$h^{\mathrm{A}}_k[1]$	$h^{\mathrm{A}}_k[i], i > 1$
$k = 0$	0	$at\_\ell_1[1] \ \wedge\ at\_\ell_0[i] \ \wedge\ tloc = i$
$k = 1$	$at\_\ell_1[1] \ \wedge\ tloc = 1$	$at\_\ell_1[1] \ \wedge\ at\_\ell_1[i] \ \wedge\ tloc = i$
$k = 2$	0	$at\_\ell_1[1] \ \wedge\ at\_\ell_2[i] \ \wedge\ tloc = i$

Computing $\boldsymbol{\delta^{\mathrm{A}}_k[i]}$: As before, we begin by computing the concrete ranking functions $\delta^{\mathrm{C}}_k[j]$. The rational for their computation is that $\delta^{\mathrm{C}}_k[j](s) = 1$ for a state s, if s is pending and there exists an r-free path leading from s to a state s on which $J^{\mathrm{C}}_k[j]$ is helpful. Thus, we compute $\delta^{\mathrm{C}}_k[j] = pend_{_C} \ \wedge\ ((\neg r)\, E\,\mathcal{U}\, h^{\mathrm{C}}_k[j])$, where $E\mathcal{U}$ is the "existential-until" CTL operator.

Having found the concrete $\delta^{\mathrm{C}}_k[j]$, we proceed to project and generalize as follows: for each $k \in [0..m]$, we let $\delta^{\mathrm{A}}_k[1] = \delta^{\mathrm{C}}_k[1][1 \mapsto 1]$ and $\delta^{\mathrm{A}}_k[i] = \delta^{\mathrm{C}}_k[3][1 \mapsto 1, 3 \mapsto i]$.

Applying this procedure to TOKEN-RING(6), we obtain the following symbolic ranking functions:

$$\begin{array}{l}
\delta^{\mathrm{A}}_0[1] = \delta^{\mathrm{A}}_2[1] : \quad 0 \\
\delta^{\mathrm{A}}_1[1] : at_\ell_1[1]
\end{array}$$

$$\left.\begin{array}{l}
\delta^{\mathrm{A}}_0[i] : at_\ell_1[1] \ \wedge\ (1 < tloc < i \ \wedge\ at_\ell_{0,1}[i] \ \vee\ tloc = i) \\
\delta^{\mathrm{A}}_1[i] : at_\ell_1[1] \ \wedge\ (1 < tloc < i \ \wedge\ at_\ell_{0,1}[i] \ \vee\ tloc = i \ \wedge\ at_\ell_1[i]) \\
\delta^{\mathrm{A}}_2[i] : at_\ell_1[1] \ \wedge\ (1 < tloc < i \ \wedge\ at_\ell_{0,1}[i] \ \vee\ tloc = i \ \wedge\ at_\ell_{1,2}[i])
\end{array}\right\} \quad \text{for } i > 1$$

3.3 Validating the Premises

Having computed internally the auxiliary constructs, it only remains to check that the six premises of rule DISTRANK are all valid, for any value of N. Here we use the small model theorem stated in Claim 1 which allows us to check their validity for all values of $N \leq N_0$ for the cutoff value of N_0 which is specified in the theorem. First, we have to ascertain that all premises have the required AE form. For auxiliary constructs of the form we have stipulated in this Section, this is straightforward. Next, we consider the value of N_0 required in each of the premises, and take the maximum.

Usually, the most complicated premise is D2 and this is the one which determines the value of N_0. For program TOKEN-RING, this premise has the form (where we renamed the quantified variables to remove any naming conflicts):

$$((\forall a.pend(a)) \ \wedge\ (\exists i, i_1 \forall j, j_1.\psi(i, i_1, j, j_1))) \quad \rightarrow \quad r' \ \vee\ (\forall c.pend(c)),$$

which is logically equivalent to

$$\forall i, i_1, c \, \exists a, j, j_1 . \Big(pend(a) \ \wedge \ \psi(i, i_1, j, j_1) \quad \rightarrow \quad r' \ \vee \ pend(c) \Big)$$

The **type**$_1$ variables which are universally quantified or appear free in this formula are $\{i, i_1, c, tloc, 1, N\}$ whose count is 6. It is therefore sufficient to take $N_0 = 6$. Having determined the size of N_0, it is straightforward to compute the premises of $S(N)$ for all $N \leq N_0$ and check that they are valid, using BDD symbolic methods.

The same form of auxiliary constructs can be used in order to automatically verify algorithm BAKERY(N), for every N. However, this requires the introduction of an auxiliary variable *xmin* into the system. Variable *xmin* is the index of the process which holds the ticket with minimal value. In a future work we will present an extension of the method which enables us to verify BAKERY with no additional auxiliary variables.

4 Cases Requiring an Existential Invariant

In some cases, assertions of the form $\forall i.u(i)$ are insufficient for capturing all the relevant features of the constructs φ^{A} and $pend^{\mathrm{A}}$, and we need to consider assertions of the form $\forall i.u(i) \ \wedge \ \exists j.e(j)$. Consider, for example, program CHANNEL-RING, presented in Fig. 3.

$$
\begin{array}{l}
\quad\quad\quad \textbf{in} \quad N \quad : \textbf{natural where } N > 1 \\
\quad\quad\quad\quad\; chan : \textbf{array}[1..N] \textbf{ of boolean where } chan[i] = (i = 2) \\
\overset{N}{\underset{i=1}{\Big|\Big|}} P[i] :: \left[\begin{array}{l} \textbf{loop forever do} \\ \left[\begin{array}{l} \ell_0 : \textbf{if } chan[i] \textbf{ then } (chan[i], chan[i \oplus_N +1]) := (0,1) \\ \quad\;\; \textbf{go to } \{\ell_0, \ell_1\} \\ \ell_1 : \textbf{await } chan[i] \\ \ell_2 : \textbf{Critical} \end{array} \right] \end{array} \right]
\end{array}
$$

Fig. 3. Program CHANNEL-RING

In this program the location of the token is identified by the index i such that $chan[i] = 1$. Computing the universal invariant according to the previous methods we obtain $\varphi^{\mathrm{A}} : \forall i.(at\_\ell_{0,1} \ \vee \ chan[i])$, which is valid but insufficient in order to establish the existence of a helpful transition for every pending state.

Using a recent extension to the invariant-generation method, we can now derive invariants of the form $\forall i.u(i) \ \wedge \ \exists j.e(j)$. Applying this method to the above example, we obtain

$$\varphi^{\mathrm{A}} : \forall i.(at\_\ell_{0,1} \ \vee \ chan[i]) \ \wedge \ \exists j.chan[j]$$

Using this extended form of an invariant for both φ^{A} and $pend^{\mathrm{A}}$, we can complete the proof of program CHANNEL-RING using the methods of Section 3.

5 Conclusion and Future Work

The paper showed how to extend the method of invisible invariants to a fully automatic proof of parameterized systems $S(N)$ for any value of N. We presented rule DISTRANK, a distributed ranking proof rule that allows an automatic computation of helpful assertions and ranking functions, which can be also automatically validated.

The current method only deals with helpful assertions which depend on a single parameter. In a future work, we plan to present a rule which enables us to deal with helpful assertions of the form $h_i : \forall j \neq i.\psi(i,j)$, and extend its applicability to systems with strong fairness requirements.

References

1. K. R. Apt and D. Kozen. Limits for automatic program verification of finite-state concurrent systems. *Information Processing Letters*, 22(6), 1986.
2. T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. Zuck. Parameterized verification with automatically computed inductive assertions. In CAV'01, pages 221–234. LNCS 2102, 2001.
3. K. Baukus, Y. Lakhnesche, and K. Stahl. Verification of parameterized protocols. *Journal of Universal Computer Science*, 7(2):141–158, 2001.
4. N. Bjørner, I. Browne, E. Chang, M. Colón, A. Kapur, Z. Manna, H. Sipma, and T. Uribe. STeP: The Stanford Temporal Prover, User's Manual. Technical Report STAN-CS-TR-95-1562, Computer Science Department, Stanford University, 1995.
5. N. Bjørner, I. Browne, and Z. Manna. Automatic generation of invariants and intermediate assertions. In 1st *Intl. Conf. on Principles and Practice of Constraint Programming*, pages 589–623. LNCS 976, 1995.
6. E. Clarke, O. Grumberg, and S. Jha. Verifying parametrized networks using abstraction and regular languages. In (CONCUR'95), pages 395–407. LNCS 962, 1995.
7. M. Colon and H. Sipma. Practical methods for proving program termination. In CAV'02, pages 442–454. LNCS 2404, 2002.
8. E. Emerson and V. Kahlon. Reducing model checking of the many to the few. In (CADE'00), pages 236–255, 2000.
9. E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In POPL'95, 1995.
10. E. Gribomont and G. Zenner. Automated verification of szymanski's algorithm. In TACAS'98, pages 424–438. LNCS 1384, 1998.
11. V. Gyuris and A. P. Sistla. On-the-fly model checking under fairness that exploits symmetry. In CAV'97. LNCS 1254, 1997.
12. B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In TACAS'00. LNCS 1785, 2000.
13. 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.
14. Y. Kesten, A. Pnueli, E. Shahar, and L. Zuck. Network invariants in action. In CONCUR'02, pages 101–105. LNCS 2421, 2002.
15. D. Lesens, N. Halbwachs, and P. Raymond. Automatic verification of parameterized linear networks of processes. In POPL'97, 1997.
16. Z. Manna and A. Pnueli. Completing the temporal picture. *Theoretical Comput. Sci.*, 83(1):97–130, 1991.
17. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, 1995.

18. K. McMillan. Verification of an implementation of Tomasulo's algorithm by compositional model checking. In CAV'98, pages 110–121. LNCS 1427, 1998.
19. A. Pnueli, S. Ruah, and L. Zuck. Automatic deductive verification with invisible invariants. In TACAS'01, pages 82–97. LNCS 2031, 2001.
20. A. Pnueli, J. Xu, and L. Zuck. Liveness with $(0, 1, \infty)$-counter abstraction. In CAV'02, pages 107–122. LNCS 2404, 2002.
21. N. Shankar, S. Owre, and J. Rushby. The PVS proof checker: A reference manual. Technical report, Comp. Sci.,Laboratory, SRI International, Menlo Park, CA, 1993.
22. L. Zuck, A. Pnueli, and Y. Kesten. Automatic verification of free choice. In *Proc. of the* $3^r d$ *workshop on Verification, Model Checking, and Abstract Interpretation*, pages 208–224. LNCS 2294, 2002.