Verifying that Invariants are Context-Inductive

Vlad Rusu e-mail: rusu@irisa.fr fax: (+33) 2 99 84 25 32

IRISA/INRIA Rennes, France

Abstract. We study the deductive verification of infinite-state systems modeled by extended automata. Typically, this process requires proving many invariants, and automatically discharging these proof obligations would save the user a significant amount of effort. To accomplish this we present techniques for automatically verifying that invariants are inductive in a given context, and identify a class of systems and a logic for expressing invariants and contexts for which the problem is decidable.

1 Introduction

Deductive techniques [15] provide powerful and general ways to verify properties of infinite-state systems, but their use in mechanical verification tools [9, 17, 4, 16] requires a serious amount of effort from the human verifier. Most of this effort is dedicated to finding and proving *invariants*, that is, properties that are true in all the reachable states of the system. Invariants are useful for proving both safety and liveness properties (see, *e.g.*, [14]). One way to prove that a property is an invariant is by showing that it is *inductive*, *i.e.*, true in the initial states of the system and preserved by all transitions. However, most invariants are not inductive by themselves, but only in a given *context*, defined by auxiliary invariants. Two main techniques exists for obtaining auxiliary invariants:

- invariant generation [2, 3, 11] consists in performing static analysis of the specification to automatically obtain relations between control and data,
- invariant strengthening [7, 12] is an interactive process that consists in repeatedly adding information to the invariant under proof until it becomes inductive. The information is extracted from previous unsuccessful proofs.

In this paper we study the deductive verification of a class of infinite-state systems called PF-*automata*, which consist of guarded commands over a finite-state structure, operating on infinite-state variables such as integers and uninterpreted functions. The class is useful for modeling, *e.g.*, programs that operate on arrays, or communication protocols with unbounded channels that carry unbounded values. Specifically, we define a method for *verifying that invariants are context-inductive* (abbreviated as VICI) that solves the following problem: given predicates $\mathcal{A}$, $\mathcal{I}$ in a certain logic, and a PF-automaton $\mathcal{P}$, is it the case that $\mathcal{I}$ is inductive over $\mathcal{P}$ in the context $\mathcal{A}$ of auxiliary invariants known of $\mathcal{P}$.

We show that the problem is decidable when $\mathcal{A}$ and $\mathcal{I}$ are expressed in a fragment of the theory of Presburger arithmetic with uninterpreted functions [10]

and $\mathcal{P}$ is a PF-automaton. Moreover, the auxiliary predicates $\mathcal{A}$ are generated automatically using invariant generation techniques.

The VICI method improves the automation of deductive verification in the following ways. First, it may be that the invariant under proof can be inferred directly from the automatically generated invariants, or is inductive in the context defined by those invariants. In both cases, the proof is automatic and the effort of interactively proving the invariant, *e.g.*, using invariant strengthening, is saved. Otherwise, even if invariant strengthening needs to be applied, the ability of automatically proving inductiveness may save effort, by detecting that the invariant under proof is inductive within fewer invariant-strengthening iterations. Finally, if the user has a good intuition of a potentially useful inductive or context-inductive *auxiliary* invariant, having an automatic procedure allows to quickly check this intuition and to fix errors in a few trial-and-error steps. We demonstrate the gains that the method provides in the context of a deductive verification using the PVS theorem prover [17] of a sliding-window protocol.

The rest of the paper is organized as follows. In Section 2 we present the theory of Presburger arithmetic with uninterpreted function symbols, and define a decidable fragment of this theory. In Section 3 we define the class of PF-automata, which are extended automata with guards and assignments in the fragment, and present the problem of verifying that invariants are context-inductive (VICI). In Section 4 we define invariant-generation techniques for PF-automata, and show that the VICI problem is decidable for the formalisms introduced in the previous sections. In Section 5 we present the verification of a sliding window protocol in PVS and how the VICI method is employed to automatically verify inductive and context-inductive invariants, which made up for 50% of all invariants proved. In Section 6 we present related work, conclusions, and future work.

2 Presburger Arithmetic with Uninterpreted Functions

In this section we briefly describe the theory of Presburger arithmetic with uninterpreted function symbols, and define a decidable fragment of this theory.

2.1 The Full Theory

Let V be a set of integer variables, and F be a set of function symbols. For each function $f \in F$, we only know its *arity* (a natural number n), and the fact that it is a function from $\mathbb{Z}^n$ to $\mathbb{Z}$. A *term with function symbols* is either a variable, or a function application to a term, or an affine combination of those. An *inequality* is a comparison $(<, >, \leq, \geq, =)$ between terms. A *quantifier-free formula* is a finite Boolean combination of inequalities. A formula of Presburger arithmetic with uninterpreted function symbols (or PF, for short) is a finite Boolean combination of inequalities, in which some variables can be quantified. Quantification over function symbols is not allowed. Thus, if x, y, z, u are variables and f, g are functions of arity one, $x + 2y + f(g(z))$ is a term, $x \leq y \wedge x + 2y + f(g(z)) > 0$ is a quantifier-free formula, and $\forall x.x \leq y + f(g(z)) \wedge \exists u.x + 2y + f(z) > g(u)$ is a PF formula. Satisfiability in PF is Σ_1^1-complete (validity is Π_1^1-complete [10]).

2.2 Decidable Fragments

However, satisfiability is decidable in the quantifier-free fragment of PF [22]. In this section we build on this result to define a larger decidable fragment.

Decidability of the quantifier-free fragments. The result [22] is based on a simple observation. In a quantifier-free formula with uninterpreted function symbols, the only relevant property about functions is that they map equals to equals, and, by instantiating this property to finitely many terms, it is possible to obtain an equivalent Presburger arithmetic formula. Let φ be a formula of the quantifier-free fragment of PF. For simplicity, we suppose that in φ there is only one unary function symbol f, which is only applied to two terms, t_1 and t_2. Then, φ is satisfiable if and only if the following Presburger formula is satisfiable:

$$\tilde{\varphi}: \quad \varphi[f(t_1)/f_1, f(t_2)/f_2] \wedge (t_1 = t_2 \supset f_1 = f_2) \tag{1}$$

That is, in $\tilde{\varphi}$, the function applications $f(t_i)$ from φ are replaced by new integer variables f_i, and the general property that function f maps equals to equals is instantiated to "equality of the terms t_i implies equality of the variables f_i". Indeed, a model for φ trivially induces a model for $\tilde{\varphi}$ by choosing $f_1 = f(t_1)$ and $f_2 = f(t_2)$. Conversely, if there exists a model for $\tilde{\varphi}$, then a model for φ can be obtained by choosing the value for f such that $f(t_1) = f_1$, $f(t_2) = f_2$, and by letting $f(i)$ be an arbitrary value at any other position than t_1, t_2.

Decidability of the existential fragment. Consider now the existential fragment of PF (*i.e.*, only existential quantifiers are allowed, and under the scope of an even number of negations). Modulo a renaming of variables, it is possible to move all quantifiers to the outermost level. Then, a formula $\exists x.\varphi$ has a model if and only if φ also has one. Indeed, if there exists values of the functions and free variables that satisfy φ, then the same values satisfy $\exists x.\varphi$, and if there exists a model for $\exists x.\varphi$, then this model, augmented with the "witness" value of x for the existential quantifier, is a model for φ. Thus, the fragment is decidable.

Decidability of the semi-universal fragment. The universal fragment of PF consists of formulas in which only universal quantifiers are allowed, under an even number of negations. This fragment is highly undecidable[1]. Nevertheless, universal formulas are useful: for example, specifying a property of *all* the elements of a parametric-sized vector requires a universal quantifier. In the remainder of this section we define a class of PF formulas with universal quantifiers for which satisfiability is decidable. This result will be used in the subsequent sections, where we define extended automata with guards and assignments in the class.

[1] This can be shown, *e.g.*, by encoding the recurrence problem for 2-counter automata.

Definition 1. *A* shielded PF formula *is a* PF *formula of the form* $\psi: \quad \forall i.\psi'$ *where* ψ' *is a quantifier-free formula with the property that function symbols* f *may only appear within terms of the form* $f(i)$. □

That is, the formula $\forall i.(f(i) > j \supset j > g(i))$ is shielded, but $\forall i.(f(i+1) = 0)$ and $\forall i.(g(i) \geq f(y))$ are not.

Also, let the *function depth* of a formula ψ be the deepest nesting of function symbols in ψ. That is, both formulas above have function depth 1.

Definition 2. *A* PF *formula* ϑ *is* semi-universal *if it is a conjunction* $\vartheta: \quad \varphi \wedge \psi_1 \wedge \ldots \wedge \psi_k$ *of function depth 1, where* φ *is quantifier-free, every* ψ_j $(j = 1, \ldots, k)$ *is shielded, and the following semantic property holds: for every model* $\mathcal{M}_\varphi$ *of* φ *there exists a model* $\mathcal{M}_\psi$ *of* $\psi: \psi_1 \wedge \ldots \wedge \psi_k$ *such that* $\mathcal{M}_\varphi$, $\mathcal{M}_\psi$ *agree on the values of the free variables that occur in both* φ, ψ. □

Example 1. Consider the following formula

$$\vartheta: \quad \underbrace{f(x+1) \leq f(y)+1}_{\varphi} \wedge \underbrace{\forall i.(f(i) = y+1)}_{\psi}$$

It is a semi-universal formula, as it satisfies the syntactic conditions of Definition 2, *i.e.* has function depth 1, φ is quantifier-free, and ψ is shielded. For the semantic condition, note that for every model $\mathcal{M}_\varphi$ (*e.g.*, Equation 2) there exists also a model $\mathcal{M}_\psi$ (*e.g.*, Equation 3) such that $\mathcal{M}_\varphi$, $\mathcal{M}_\psi$ agree on the value of y, *i.e.*, the free variable that occurs in both φ and ψ.

$$\mathcal{M}_\varphi: \quad \{x = 0, y = 2, f(1) = 3, f(2) = 3, \text{and for } k \notin \{1,2\}: \ f(k) = 0\} \tag{2}$$
$$\mathcal{M}_\psi: \quad \{y = 2, \text{for all } k: \ f(k) = 3\} \tag{3}$$

□

Definition 2 may be hard to use in practice because of the semantic property that is involved. However, it turns out to be the exact definition we need in the following sections. A sufficient (and easier to check) condition for a formula to be semi-universal is presented at the end of this section.

Lemma 1. *Satisfiability in the class of semi-universal formulas is decidable.*

A complete proof of this lemma can be found in [19]. Here, we just give the main idea and illustrate it through an example. To decide a semi-universal formula $\vartheta: \varphi \wedge \psi$ where $\psi: \forall i_1 \psi_1 \wedge \ldots \forall i_k . \wedge \psi_k$, the idea is to instantiate every quantifier in ψ to every term to which a function symbol is applied in φ. For example, when ψ is of the form $\forall i.\psi_1$ we obtain the following formula (4), which is equivalent to ϑ

$$\vartheta': \quad \underbrace{\varphi \wedge \bigwedge_{i \in \{t_1, \ldots, t_m\}} \psi_1(i, f_1(i), \ldots, f_n(i))}_{\varphi'} \wedge \underbrace{\forall i \notin \{t_1, \ldots, t_m\}.\psi_1(i, f_1(i), \ldots, f_n(i))}_{\psi'} \tag{4}$$

Then, the whole formula ϑ' is satisfiable if and only if φ' is satisfiable: if φ' has a model $\mathcal{M}_{\varphi'}$, then there exists also a model $\mathcal{M}_{\psi'}$ for ψ' that agrees with $\mathcal{M}_{\varphi'}$ on the values of the variables common to both formulas φ', ψ', and using this fact it is possible to construct a model for θ' from $\mathcal{M}_{\varphi'}$, $\mathcal{M}_{\psi'}$ as follows. The values of the free variables that appear in one of the formulas φ', ψ' but not in the other are chosen from that formula's model, and the values of the free variables that appear in both formulas are chosen from either model. The values of the functions are defined as follows: at all positions i equal to the value of some term t_k $(k = 1, \ldots, m)$ in $\mathcal{M}_{\varphi'}$, we evaluate $f_j(i)$ $(j = 1, \ldots, n)$ according to $\mathcal{M}_{\varphi'}$, and at all remaining positions, we evaluate $f_j(i)$ according to $\mathcal{M}_{\psi'}$. It is not hard to show that this valuation constitutes a model for (4), thus, for ϑ.

Example 2. Consider the semi-universal formula ϑ from Example 1. The formula ϑ is equivalently rewritten as:

$$\vartheta' : \quad \underbrace{f(x+1) \leq f(y)+1 \wedge f(x+1) = y+1 \wedge f(y) = y+1}_{\varphi'} \wedge \underbrace{\forall i \notin \{x+1, y\}.(f(i) = y+1)}_{\psi'} \tag{5}$$

Let $\mathcal{M}_{\varphi'} : \quad \{x = 0, y = 2, f(1) = 3, f(2) = 3, \text{and for all } k \notin \{1, 2\} : f(k) = 0\}$ be a model for φ'. Here, $\mathcal{M}_{\varphi'}$ is also a model for φ. A corresponding model for ψ, which agrees with $\mathcal{M}_{\varphi'}$ on the values of the common free variables, is $\mathcal{M}_{\psi} : \quad \{y = 2, \text{for all } k : f(k) = 3\}$ We extend $\mathcal{M}_{\psi}$ into a model for $\mathcal{M}_{\psi'}$ of ψ' by choosing $x = 0$ as in $\mathcal{M}_{\varphi'}$, and build a valuation $\mathcal{M}_{\vartheta'}$ as follows:

The values of free variables x and y are chosen from $\mathcal{M}_{\varphi'}$: $x = 0$, $y = 2$. The values of $f(i)$ such that $i = x + 1$ or $i = y$, that is, for $i \in \{1, 2\}$, are also chosen from $\mathcal{M}_{\varphi'}$: $f(1) = 3,\ f(2) = 3$. The values of $f(i)$ such that $i \notin \{1, 2\}$ are chosen from $\mathcal{M}_{\psi'}$, that is, for all $i \notin \{1, 2\} : \ f(i) = 3$. Thus, the valuation $\mathcal{M}_{\vartheta'}$, which is also a model for ϑ, is $\mathcal{M}_{\vartheta'} : \quad \{x = 0, y = 2, \text{for all } k : f(k) = 3\}$. □

We now give a simple condition for a formula to be semi-universal.

Definition 3. *The* universal closure *of a formula ψ is obtained by universally quantifying every free variable in ψ. A formula is* universally satisfiable *if its universal closure is satisfiable.* □

Definition 4. *A* simple semi-universal formula *is a conjunction $\varphi \wedge \forall i_1.\psi_1 \wedge \ldots \wedge \forall i_k.\psi_k$ satisfying the syntactic conditions of Definition 2 and the semantic condition that $\psi : \forall i_1.\psi_1 \wedge \ldots \wedge \forall i_k.\psi_k$ is universally satisfiable.*

In a simple semi-universal formula, there exist values of the functions that, together with any values of the variables, constitute a model for the quantified part ψ. In particular, the values of the variables can be chosen from a model $\mathcal{M}$ of the unquantified part φ, thus, the semantic constraints from Definition 2 are met. Checking Definition 4 is easier than checking Definition 2. It can be done, *e.g.,* using theorem proving by providing witness values for the functions.

Example 3. Consider the formula $\vartheta : \quad \underbrace{f(x+1) \leq f(y)+1}_{\varphi} \wedge \underbrace{\forall i.(f(i) = i+1)}_{\psi}$

It is a simple semi-universal formula as it satisfies the conditions of Definition 4, *i.e.* has functions depth 1, φ is quantifier-free, and ψ is shielded; for the semantic condition, note that ψ coincides with its universal closure and is satisfiable. □

3 PF-Automata

In this section we define the syntax and semantics of a class of extended automata with guards and assignments in the semi-universal fragment of PF.

Definition 5. *A* variable assignment *to variable x is an expression of the form $x' = t$, where t is a term of* PF. □

Definition 6. *A* function assignment *to function f is a shielded* PF *formula of function of depth 1, which has the form $\forall i.(e_1(i) \supset f'(i) = e_2(i))$, where e_1 is a quantifier-free formula, e_2 is a term, and f' does not occur in e_1, e_2.* □

Definition 7. *A* PF-automaton *is a tuple $\langle Q, q^0, V, F, \Theta, \mathcal{T} \rangle$:*

- *$Q = \{1, 2, \ldots, |Q|\}$ is a finite set of locations,*
- *$q^0 \in Q$ is the initial location,*
- *V is a finite set of integer variables,*
- *F is a finite set of unary function symbols,*
- *Θ is a simple semi-universal* PF *formula, called the initial condition,*
- *$\mathcal{T}$ is a finite set of transitions. Each transition is a tuple $\langle q, \gamma, \nu, \phi, q' \rangle$ where*
  - *$q \in Q$ is the origin of the transition,*
  - *γ is a quantifier-free* PF *formula of function depth at most 1, called the guard of the transition,*
  - *ν is a finite set of variable assignments (cf. Definition 5). For each variable $v \in V$, there is at most one assignment to v in ν,*
  - *ϕ is a finite set of function assignments (cf. Definition 6). For each function $f \in F$, there is at most one assignment to f in ϕ,*
  - *$q' \in Q$ is a location called the destination of the transition.* □

Note that the initial condition Θ is required to be a simple semi-universal formula. As membership in this class is not decidable, other techniques (*e.g.,* theorem proving, *cf.* end of Section 2.2) may be needed to establish that a given structure is a PF-automaton. We expect that PF-automata that model real programs have rather simple initial conditions, whose satisfiability is not hard to assess.

PF-automata are useful for modeling programs with unbounded data structures such as files, buffers, and arrays of parametric size. In Definition 7, we have assumed that the only basic type is integer, but, of course, other ground types (Booleans, enumerations, records, subranges) can be encoded using integers. The restriction that there is at most one assignment for each variable and function application is useful for avoiding semantic complications (*i.e.,* situations where a function gets two different values simultaneously). It can be dealt with in practice by introducing new transitions to sequentialize the assignments.

Figure 1 is an example of PF-automaton, which models the sorting algorithm of a vector g of parametric size m using a bubble-sorting procedure. Initially, the actual parameters g and m are equal to the formal parameters of the procedure, f and n. Then, f is sorted and copied back into g at the end of the procedure.

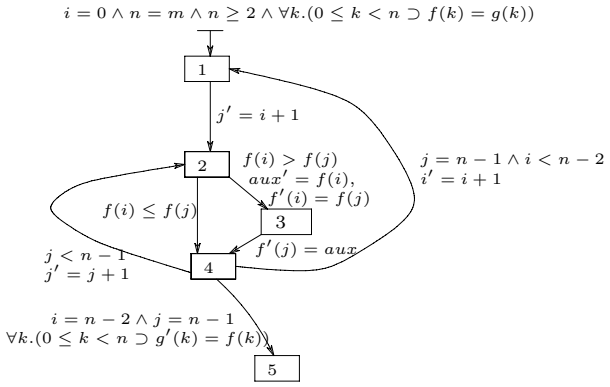

Fig. 1. Example of PF-automaton: Vector Sorting

In Figure 1, expressions such as $f'(i) = f(j)$ are abbreviations for function assignments of the form, *e.g.,* $\forall k.(k = i \supset f'(k) = f(j))$. The meaning of such an assignment is that the next value of $f(i)$ will be current value of $f(j)$, and the next value of $f(w)$ for $w \neq j$ will remain equal to its current value. By convention, variables that do not appear in assignments also remain unchanged.

A simple, yet key invariant for proving the correctness of the sorting procedure is $pc = 4 \supset f(i) \leq f(j)$, that is, whenever control is at location 4, the elements $f(i)$ and $f(j)$ have been properly sorted. We show in the next section how to automatically prove such invariants.

Semantics of PF-automata. A *valuation* is a mapping that assigns, to each free variable appearing in the automaton, a value in $\mathbb{Z}$, and to each function symbol, a function from $\mathbb{Z}$ to $\mathbb{Z}$. We denote by $\mathcal{V}$ the set of all valuations. A *state* is a pair (q, v) consisting of a location $q \in Q$ and a valuation $v \in \mathcal{V}$. Note that, for a PF-automaton with at least one function symbol, there is an uncountably infinite set of states. An *initial state* is a state of the form (q^0, v^0) such that $v^0 \models \Theta$, that is, the location is initial and the values of the variables and functions satisfy the initial condition Θ. The set of states is denoted by $\mathcal{S}$, and the set of initial states is denoted by $\mathcal{S}^0$. Each transition $\tau \in \mathcal{T}$ defines a *transition relation* $\varrho_\tau \subseteq \mathcal{S} \times \mathcal{S}$, in the following way. Intuitively, s and s' are in the relation ϱ_τ if the location of s (*resp.* of s') is the origin (*resp.* destination) of τ, and the variables and functions in s satisfy the guard γ of τ. Moreover, the variables and functions get new values according to the assignments of τ. A formal definition of the ϱ_τ transition relation can be found in [19].

We denote the global transition relation of the PF-automaton by $\varrho = \bigcup_{\tau \in \mathcal{T}} \varrho_\tau$. A *run* is a sequence of states $\rho : \quad s_0, s_1, \ldots, s_n$ such that $s_0 \in \mathcal{S}^0$, and for $i = 0, \ldots, n-1$, $\varrho(s_i, s_{i+1})$ holds. A state predicate $\mathcal{I}$ is an *invariant* if it holds at every state of every run. A predicate $\mathcal{I}$ is *inductive* if it holds at all initial states and for all states s, s', $\mathcal{I}(s)$ and $\varrho(s, s')$ imply $\mathcal{I}(s')$. For $\mathcal{I}$ and $\mathcal{A}$ state

predicates, $\mathcal{I}$ is *inductive in the context* $\mathcal{A}$ if $\mathcal{I}$ holds initially and for all states s, s', $\mathcal{I}(s)$ and $\mathcal{A}(s)$ and $\varrho(s, s')$ imply $\mathcal{I}(s')$. Note that an inductive predicate is an invariant, and if $\mathcal{A}$ is an invariant and $\mathcal{I}$ is inductive in the context $\mathcal{A}$, then $\mathcal{I}$ is also an invariant. The problem of *verifying that invariants are context-inductive* is: given a predicate $\mathcal{I}$ and an invariant $\mathcal{A}$, is $\mathcal{I}$ inductive in the context $\mathcal{A}$.

4 VICI: Verifying that Invariants are Context-Inductive

In this section we provide techniques for automatically generating auxiliary invariants for PF-automata, and for automatically proving the inductiveness for a class of predicates in the context of the generated invariants.

For a transition τ and an arbitrary state predicate Ψ, the predicate $post_\tau(\Psi)$ characterizes the states s that can be reached by taking transition τ from a state $s\_$ satisfying Ψ :

$$post_\tau(\Psi): \quad \exists s\_.\varrho_\tau(s\_, s) \wedge \Psi(s\_)$$

This operator is also defined globally: $post(\Psi)$: $\exists s\_.\varrho(s\_, s) \wedge \Psi(s\_)$ or, equivalently, $post(\Psi)$: $\bigvee_{\tau\in\mathcal{T}} post_\tau(\Psi)$. We will need in the sequel the symbolic form of $post_\tau(\Psi)$ for $\tau = \langle q\_, \gamma, \nu, \phi, q\rangle$ a transition of the PF-automaton. Let $\overline{x}$ denote the variables of the PF-automaton. The presence at a given location l can be encoded, using a new integer variable pc called the *control variable*, by the constraint $pc = l$. Without restricting the generality, we assume the automaton has only one function symbol f, thus, the function assignments ϕ of transition τ consists of one element, of the form $\forall i.(e_1(i) \supset f'(i) = e_2(i))$. Then, $post_\tau(\Psi)$ can be written as formula (6) with free variables $\overline{x}$, pc, and function f:

$$\begin{aligned}\exists f\_\exists\overline{x}\_.(\ pc = q \wedge \gamma(\overline{x}\_, f\_) \wedge \overline{x} = \nu(\overline{x}\_, f\_) \wedge \Psi(\overline{x}\_, f\_) \wedge \qquad (6)\\ \forall i\ .(e_1(i, \overline{x}\_, f\_) \supset f(i) = e_2(i, \overline{x}\_, f\_) \wedge \forall i.(\neg e_1(i, \overline{x}\_, f\_) \supset f(i) = f\_(i)))\end{aligned}$$

That is, the next control is at location q, and Ψ, γ must hold at the previous values of the variables $\overline{x}\_$ and function $f\_$. Moreover, the variables are modified by the assignments ν, depending on their previous values and that of the function, and the function is updated at all positions i where e_1 holds of i, $\overline{x}\_$, and $f\_$.

Note that Formula (6) is not a PF formula, because it quantifies over function symbols. However, we will be interested in the *satisfiability* of such formulas, for which existential quantification does not matter. This same argument was used in Section 2.2 for showing the decidability of the existential fragment of PF.

4.1 Invariant Generation for PF-automata

Local invariants. Let a *local invariant at location* q be any predicate $\mathcal{I}$ such that $pc = q \supset \mathcal{I}$ is an invariant. The local invariant generation techniques are based on the following observations. First, $pc = q$ is a local invariant at any location q. Second, let q be a location, $\{\tau_1, \ldots, \tau_n\}$ be the set of transitions with destination q, $\{q_1, \ldots, q_n\}$ be the set of origins of those transitions, and

$\mathcal{I}_1, \ldots \mathcal{I}_n$ be local invariants at locations $\{q_1, \ldots, q_n\}$, respectively. Then, it is not hard to show that the predicate $pc = q \wedge \bigvee_{i=1}^{n} post_{\tau_i}(\mathcal{I}_i)$ is a local invariant at location q.

We now describe the general form of invariants obtained using these techniques. For this, note from the symbolic form (6) of *post* that $post_{\tau_i}(pc = q_i)$ can be written as $pc = q \wedge \exists f\_ \exists \overline{x}\_ . \Gamma'(\overline{x}\_, f\_, \overline{x}, f)$, where Γ' is a formula only on variables $\overline{x}\_, f\_, \overline{x}, f$ in which the only quantifiers are the ones that come from the function assignments. By instantiating these quantifiers to *the terms to which f and $f\_$ are applied in the quantifier-free part of Γ'*, just as in the proof of Lemma 1, we obtain a formula $pc = q \wedge \exists f\_ \exists \overline{x}\_ . \Gamma(\overline{x}\_, f\_, \overline{x}, f)$, where Γ is quantifier-free.

Thus, starting from the trivial local invariants $pc = q_i$ at locations q_i, we obtain at location q a local invariant of the form $pc = q \wedge \bigvee_{i=1}^{n} \exists f\_ \exists \overline{x}\_ . \Gamma_i(\overline{x}\_, f\_, \overline{x}, f)$ such that each Γ_i is quantifier-free. By iterating this process, taking into account the fact that, modulo a renaming of variables, the existential quantifiers can always be brought to the outermost level, we obtain that invariant generation produces local invariants of the form (7), where Γ is quantifier-free:

$$pc = q \wedge \exists f\_, f_{2\_}, \ldots, f_{k\_} . \exists \overline{x}\_, \overline{x}_{2\_}, \ldots, \overline{x}_{k\_} . \Gamma(f, f\_, f_{2\_}, \ldots, f_{k\_}, \overline{x}, \overline{x}\_, \overline{x}_{2\_}, \ldots, \overline{x}_{k\_}) \tag{7}$$

For example, consider location 3 of the PF-automaton depicted in Fig 1. Starting from the trivial local invariant $pc = 2$ at location 2, we obtain, after some arithmetic and logical simplifications, the local invariant at location 3:

$$\exists f\_ . (f\_(i) > f\_(j) \wedge aux = f\_(i) \wedge f(i) = f\_(j) \wedge (j \neq i \supset f(j) = f\_(j)))$$

Now, using the above invariant and again the trivial local invariant $pc = 2$ at location 2, we obtain at location 4 after some simplification the local invariant

$$\exists f\_, f_{2\_}, aux\_ . (f(i) \leq f(j) \vee (f_{2\_}(i) > f_{2\_}(j) \wedge aux\_ = f_{2\_}(i) \wedge f\_(i) = f_{2\_}(j) \wedge \tag{8}$$
$$(j \neq i \supset f\_(j) = f_{2\_}(j)) \wedge (i \neq j \supset f(i) = f\_(i)) \wedge f(j) = aux\_ \wedge aux = aux\_))$$

Global invariants. This second technique consists in projecting the PF-automaton onto its integer variables and applying linear relation analysis [11] to the resulting automaton. The technique is global in that it works not just by examining a location and the incoming transitions (as local invariant generation does), but by computing an over-approximation of the set of reachable states. However, linear relation analysis works only on extended automata with scalar types, and to apply it we first need to project a PF-automaton on its integer variables.

Combining local and global invariants: an example. By applying the global invariant generation technique, we obtain at location 4 the invariant $i < j$. The latter, in conjunction with (8) implies $f(i) \leq f(j)$, which is the local invariant at location 4 that we wanted to prove (cf. Section 3). This is an example of proving an invariant only by using automatically generated auxiliary invariants. In the next section, we show how to automate the proofs of such implications, as well as how to automate the more powerful technique of proving that predicates are *inductive* in a given context.

4.2 Solving the VICI Problem

The VICI problem (cf. end of Section 3) amounts to the following: given a state predicate $\mathcal{I}$ and an invariant $\mathcal{A}$, decide the validity of all formulas $pc = q^0 \wedge \Theta \supset \mathcal{I}$ and $post_\tau(\mathcal{A} \wedge \mathcal{I}) \supset \mathcal{I}$, for all transitions τ of the PF-automaton. That is, decide if $\mathcal{I}$ holds initially and if, by knowing $\mathcal{A}$, one can infer that any state reachable from $\mathcal{I}$ by some transition is still in $\mathcal{I}$. If this is the case, then $\mathcal{I}$ is clearly an invariant. Equivalently, solving VICI is done by showing that none of the formulas

$$pc = q^0 \wedge \Theta \wedge \neg\mathcal{I} \tag{9}$$

and, for τ an arbitrary transition of the PF-automaton

$$post_\tau(\mathcal{A} \wedge \mathcal{I}) \wedge \neg\mathcal{I} \tag{10}$$

is satisfiable. We show how to decide satisfiability for such formulas when $\mathcal{I}$ is a semi-universal formula (Definition 2), and $\mathcal{A}$ is an invariant obtained by the techniques described in Section 4.1. The proofs rely on the fact that satisfiability for semi-universal formulas (Definition 2) is decidable.

Showing that Formula (9) is semi-universal. By the Definition 7 of PF-automata, the initial condition is simple semi-universal, thus, $\Theta : \Theta_0 \wedge \forall j_1.\Theta_1 \wedge \ldots \wedge \forall j_m.\Theta_m$, where $\Theta_0, \ldots \Theta_m$ are quantifier-free, and $\forall j_1.\Theta_1 \wedge \ldots \wedge \forall j_m.\Theta_m$ is universally satisfiable. Since $\mathcal{I}$ is also a semi-universal formula, $\mathcal{I}$ can also be written $\mathcal{I} : \mathcal{I}_0 \wedge \forall i_1.\mathcal{I}_1 \wedge \ldots \wedge \forall i_n.\mathcal{I}_n$, where $\mathcal{I}_0, \ldots \mathcal{I}_n$ are quantifier-free.

Then, Formula (9) can be written as $pc = q^0 \wedge \Theta_0 \wedge \forall j_1.\Theta_1 \wedge \ldots \wedge \forall j_m.\Theta_m \wedge \neg(\mathcal{I}_0 \wedge \forall i_1.\mathcal{I}_1 \wedge \ldots \wedge \forall i_n.\mathcal{I}_n)$. By pushing the negation inwards, and propagating the resulting existential quantifiers to the outermost level, we obtain the equivalent $\exists i_1 \ldots i_n.(pc = q^0 \wedge \Theta_0 \wedge \forall j_1.\Theta_1 \wedge \ldots \wedge \forall j_m.\Theta_m \wedge (\neg\mathcal{I}_0 \vee \neg\mathcal{I}_1 \vee \ldots \vee \neg\mathcal{I}_n))$.

For satisfiability, the existential quantifiers do not matter, therefore, we consider the above formula from which the existential quantifiers have been removed, and prove that it is semi-universal, with the quantifier-free part $\varphi : \quad pc = q^0 \wedge \Theta_0 \wedge (\neg\mathcal{I}_0 \vee \neg\mathcal{I}_1 \vee \ldots \vee \neg\mathcal{I}_n)$ and quantified part $\psi : \quad \forall j_1.\Theta_1 \wedge \ldots \wedge \forall j_m.\Theta_m$.

For this, we show that if there is a model $\mathcal{M}$ for φ, then there is also a model $\mathcal{M}'$ for ψ such that the values of the free variables in $\mathcal{M}$ and $\mathcal{M}'$ coincide. But this is just a consequence of ψ being universally satisfiable: if there exists a model $\mathcal{M}$ of φ, we build $\mathcal{M}'$ by taking the values of the free variables from $\mathcal{M}$ and the values of the functions from a model of the universal closure of ψ.

Showing that Formula (10) is semi-universal. Without restricting the generality, we assume that the PF-automaton has only one function symbol f, and that $\mathcal{I}$ in (10) is of the form $\mathcal{I} : \quad \mathcal{I}_0 \wedge \forall i_1.\mathcal{I}_1$, that is, the quantified part of $\mathcal{I}$ consists only of one conjunct. Then, using Formulas (6) and (7), Formula (10) becomes

$$\begin{aligned}\exists f_\_, f_{2\_} \ldots f_{k\_}, \overline{x}_\_, \overline{x}_{2\_} \ldots \overline{x}_{k\_}.[\ & \Gamma\ (f_\_, f_{2\_}, \ldots, f_{k\_}, \overline{x}_\_, \overline{x}_{2\_}, \ldots, \overline{x}_{k\_}) \wedge pc = q \wedge \overline{x} = \\ & \gamma\ (\overline{x}_\_, f_\_) \wedge \nu(\overline{x}_\_, f_\_) \wedge \mathcal{I}_0(\overline{x}_\_, f_\_) \wedge \forall i_1.\mathcal{I}_1(i_1, \overline{x}_\_, f_\_) \wedge \\ & \forall\ i.(e_1(i, \overline{x}_\_, f_\_) \supset f(i) = e_2(i, \overline{x}_\_, f_\_) \wedge (\neg e_1(i, \overline{x}_\_, f_\_) \\ & \supset f(i) = f_\_(i))] \wedge \exists i_1.(\neg\mathcal{I}_0(\overline{x}, f) \vee \neg\mathcal{I}_1(i_1, \overline{x}, f))\end{aligned} \tag{11}$$

Existential quantifiers are irrelevant for satisfiability, so we may remove them. In the resulting formula, the only remaining quantifiers are universal. We show that this formula is semi-universal. For this, assume that there exists a model $\mathcal{M}$ for the part of (11) outside the scope of universal quantifiers. In particular, $\mathcal{M}$ gives values to the variables $\overline{x}\_$, which we denote by $v_{\mathcal{M}}(\overline{x}\_)$. Since we have assumed $\mathcal{I}$ is semi-universal, this means that there exists a value for $f\_$, denoted $v(f\_)$, such that $\mathcal{M}' : \quad (v_{\mathcal{M}}(\overline{x}\_), v(f\_))$ constitutes a model for $\forall i_1.\mathcal{I}_1$.

Finally, we show how to extend $\mathcal{M}'$ to a model for the universally quantified part of Formula (11). For this, we just have to give a value $v(f)$ to function f. Clearly, choosing $v(f)$ such that for all $i_0 \in \mathbb{Z}$, $v(f)(i_0)$ is equal to $e_2(i/i_0, \overline{x}/v_{\mathcal{M}}(\overline{x}\_), f\_/v(f\_))$ if $e_1(i/i_0, \overline{x}/v_{\mathcal{M}}(\overline{x}\_), f\_/v(f\_))$ holds, and $v(f)(i_0)$ equals $v(f\_)(i_0)$ otherwise, will satisfy the whole quantified part of Formula (11). Here, *e.g.*, $e_1(i/i_0, \dots)$ denotes replacing i by i_0 in e_1, etc, and the proof is done.

5 Application: a Sliding Window Protocol

We present an example of deductive verification using PVS and VICI. The main verification effort is performed in PVS, and VICI is employed to prove that some invariants are inductive or context-inductive. The sliding window protocol is a well-known case study for verification methods, both deductive and algorithmic. A recent paper with complete references is [23].

The Sliding Window Protocol. The architecture of the protocol is represented in Figure 2. There are two entities, Sender and Receiver, communicating through unreliable channels that may lose data. The sender obtains data from a FIFO stream of data called the `Source`. Each data element is first saved into the sender's window, a FIFO buffer called `sndWindow`. Then, the sender takes each data element from its window, associates an index to it (a natural number that keeps track of the order in which data has been obtained from the source), and sends the resulting record called a Message to the Message Channel `MsgChan`. The latter is a lossy FIFO, which may lose, but not reorder or create messages.

On the receiver side, messages are removed from `MsgChan` and, if a message's index is within the bounds of the receiver's window `rcvWindow`, it is stored there, otherwise, it is discarded. The receiver delivers contiguous sequences of messages to the external data `Target`, and acknowledges the highest-indexed message that it has delivered, by sending that index to the acknowledgment channel `AckChan`, which is also a lossy FIFO queue. Finally, the sender reads acknowledgments from `AckChan` and, depending on what it has read, it retransmits part of its window and removes acknowledged data, leaving space for data from the data source.

Specifying the protocol in PVS. The protocol is specified as an extended automaton encoded in a PVS theory (cf. [20] for details). The *state* of the automaton is encoded as a PVS record type, with fields for all state variables for the sender, receiver, and channels. All the channels `source`, `target`, `ackChan` `msgChan` and

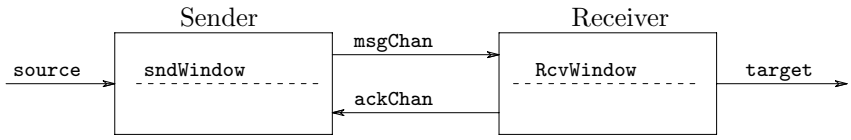

Fig. 2. Architecture of the Sliding Window Protocol

windows `sndWindow`, `rcvWindow` are modeled as uninterpreted functions. The whole protocol is translated into a PF-automaton in a straightforward manner.

Verifying properties, and how VICI increases automation. The main safety property required from the protocol is that the sequence of data delivered to the data target is a prefix of the sequence obtained from the data source.

```
main_safety_property: THEOREM invariant(LAMBDA (s: State):
        FORALL (i: nat): (i < s‘rcvLow IMPLIES s‘target(i) = s‘source(i)))
```

The verification consists in strengthening the `main_safety_property` in order to obtain an inductive invariant. This requires a number of auxiliary invariants to be proved. For example, the following lemmas describe such auxiliary invariants: they say essentially that the data source and the sender's window (including all past content) are identical up to some indexes, and the indexes are also equal:

```
source_equals_send : LEMMA invariant(LAMBDA (s: State) :
FORALL (i : nat) : i < s‘sndWinIndex IMPLIES s‘source(i) = s‘sndWindow(i))
source_equals_send_1: LEMMA invariant(LAMBDA(s: State):
                                          s‘sndWinIndex=s‘sourceIndex)
```

With VICI the second lemma is generated automatically and the first lemma is automatically proved (*i.e.*, the first predicate is inductive in the context of the second one). By contrast, using PVS amounts to performing two invariant-strengthening steps, *i.e.*, running a proof strategy, failing, examining the results, then running the strategy again. This requires some amount of user intervention.

Of a total of eleven invariants needed for proving the `main_safety_property`, three are inductive and are generated automatically using our techniques, and two are inductive in the context of the former. We have also verified a liveness property of the protocol, which required proving three auxiliary invariants, two of which are inductive. Thus, half of all auxiliary invariants were proved automatically using VICI, the remaining ones were proved interactively using PVS.

6 Conclusion, Related Work, and Future Work

We have present an approach called VICI to enhance the automation of deductive verification for a class of systems called PF-automata. The latter are extended automata with guards and assignments in a decidable fragment of the theory of Presburger arithmetic with uninterpreted function symbols, allowing, with

some restrictions, existential and universal quantifiers. The approach consists in automatically generating auxiliary invariants, and automatically verifying that candidate invariants are inductive in the context of the auxiliary invariants generated. The candidate invariants are also expressed in the same fragment of the theory of Presburger arithmetic with uninterpreted function symbols. The approach can be employed to verify programs that operate on arrays, or communication protocols with unbounded channels that carry unbounded values. Two simple examples, a bubble-sort algorithm and a sliding window protocol, are used to demonstrate the potential usefulness of the VICI approach.

Related Work. The integration of automatic, algorithmic methods into deductive techniques has received quite a lot of attention recently. Invariant generation [3, 2], abstraction [1, 8] and ways to combine them within theorem proving have been proposed [12, 21, 18]. A class of automata with uninterpreted function symbols is studied in [5], where simulation between automata in the class is shown decidable. The only results we are aware of about automata extended with Presburger arithmetic and function symbols are reported in [13]. Here, the Omega tool is used to analyze graphs whose edges are annotated by quantifier-free formulas in the logic, to compute, *e.g.*, a superset of the reachable states.

Future Work. The main ongoing work is implementing VICI and applying it to case studies. The implementation is based on the ICS tool [6] from the PVS group at SRI International's Computer Science Laboratory, an efficient decision procedure for (among other theories) the quantifier-free fragment of Presburger arithmetic with uninterpreted functions.

The VICI toolset is represented in Figure 3. (Currently, only the core of VICI is implemented, cf. Fig. 3.) Dashed lines represent modules that we are implementing, as opposed to solid lines, which representing existing software. The toolset consists of translators between PVS and ICS formats of the specification under proof and of the properties, a symbolic simulator, and an invariant generator. The symbolic simulator described in more detail in [19] can be used to debug the specification and to verify safety properties (it is a semi-decision procedure that may invalidate safety properties, but cannot prove them). Once this debugging is done, most of the verification effort happens at the PVS level, where the user performs invariant strengthening, which may further modify (debug) the specification and generates new candidate invariants. The core of VICI implements the techniques described in Section 4.2. The user may call it on candidate invariants to check their inductiveness. However, the invariants not in the scope of VICI still have to be proved interactively in PVS. More case studies are needed to assess the usefulness of VICI for automating deductive verification.

Acknowledgment. Duncan Clarke and Elena Zinovieva are implementing VICI.

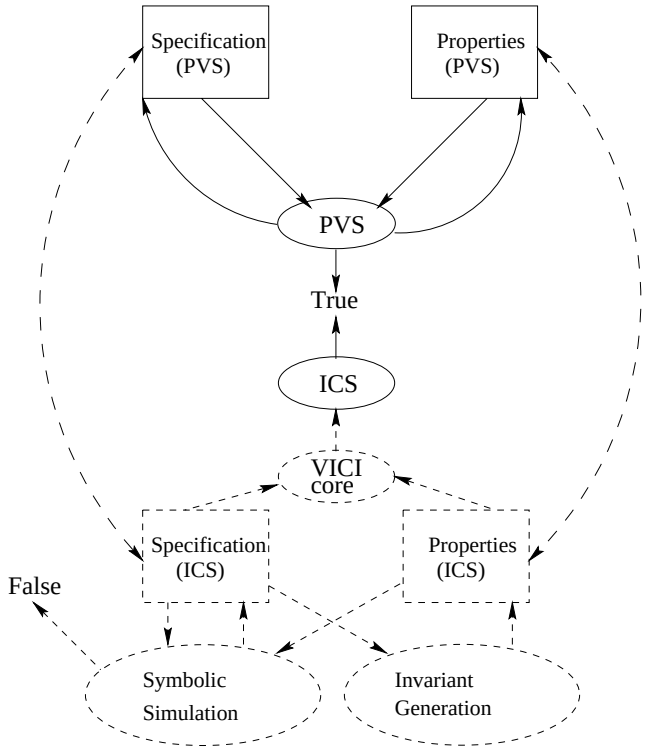

Fig. 3. The VICI toolset.

References

1. S. Bensalem, Y. Lakhnech, and S. Owre. Constructing abstractions of infinite state systems compositionally and automatically. In *Conference on Computer-Aided Verification*, LNCS 1427, 1998.
2. S. Bensalem and Y. Lakhnech. Automatic generation of invariants. *Formal Methods in System Design*, 15(1):75–92, 1999.
3. N. Bjorner, Anca. Browne and Z. Manna. *Automatic generation of invariants and intermediate assertions.* Theoretical Computer Science, 173(1):49-87, 1997.
4. C. Cornes, J. Courant, J.-C. Filliâtre, G. Huet, P. Manoury, C. Paulin-Mohring, C. Muñoz, C. Murthy, C. Parent, A. Saïbi, and B. Werner. The Coq Proof Assistant Reference Manual Version 6.1. Technical Report RT-0203, INRIA, July 1997.
5. W. Damm, A. Pnueli and S. Ruah. Herbrand automata for hardware verification. *Conference on Concurrency Theory*, LNCS 1466, 1998.
6. J.-C. Filliâtre, S. Owre, H. Rueß. and N .Shankar. *ICS: Integrated Canonizer and Solver*, To be presented at Computer-aided Verification (CAV'2001).

7. S. Graf and H. Saidi. Verifying invariants using theorem proving. In *Conference on Computer-Aided Verification*, LNCS 1102, 1996.
8. S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *Conference on Computer-Aided Verification*, LNCS 1254, 1997.
9. M. Gordon and T.F. Melham. *Introduction to the HOL system*. Cambridge University press, 1994.
10. J. Halpern. Presburger arithmetic with uninterpreted function symbols is Π_1^1-complete. *Journal of Symbolic Logic*, 56:637–642, 1991.
11. N. Halbwachs, Y.E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, 1997.
12. K. Havelund and N. Shankar. Experiments in theorem proving and model checking for protocol verification. In *Formal Methods Europe*, LNCS 1051, 1996.
13. W. Kelly, V. Maslov, W. Pugh, E. Rosser, T. Shpiesman, and D. Wonnacott. The Omega library interface guide. Available at `www.cs.umd.edu/projects/omega`.
14. Z. Manna and A. Pnueli. Tools and rules for the practicing verifier. In *CMU Computer Science: A 25th Anniversary Commemorative* ACM Press and Addison-Wesley, 1991
15. Z. Manna and A. Pnueli. *Temporal verification of reactive systems.* Vol. 1: Specification, Vol. 2: Safety. Springer-Verlag, 1991 and 1995.
16. Z. Manna and the STeP group. STeP: deductive-algorithmic verification of reactive and real-time systems. In *Computer-Aided Verification*, LNCS 1102, 1996.
17. S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107-125, 1995.
18. V. Rusu and E. Singerman. On proving safety properties by integrating static analysis, theorem proving and abstraction. In *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1579, 1999
19. V. Rusu and E. Zinovieva. Analyzing automata with Presburger arithmetic and uninterpreted function symbols. In *ICALP'01 Workshop: Verification of Parameterized Systems (VEPAS'01)*, to appear in Electronic Notes in Theoretical Computer Science 50(4), 2001. Currently available at `http://www.irisa.fr/pampa/perso/rusu/vepas`.
20. V. Rusu. Verifying a sliding window protocol using PVS. To appear in *Formal Techniques for Newtworked and Distributed Systems (FORTE'01)*, currently available at `http://www.irisa.fr/pampa/perso/rusu/forte`.
21. H. Saïdi and N. Shankar. Abstract and model check while you prove. In *Conference on Computer-Aided Verification*, LNCS 1633, 1999.
22. R. Shostak. A practical decision procedure for arithmetic with uninterpreted function symbols. Journal of the ACM, 26(2):351–360, 1979.
23. M. Smith and N. Klarlund. Verification of a sliding window protocol using IOA and MONA. In *Formal Description Techniques & Protocol Specification, Testing and Verification*, Kluwer Academic Publishing, 2000.